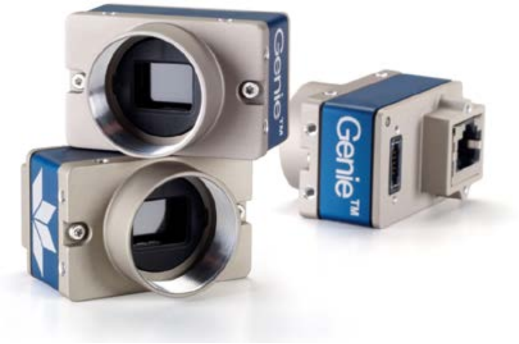
GigE-V Framework for Linux 32/64-Bit

Programmer's Manual

sensors | cameras | frame grabbers | processors | **software** | vision solutions



September 12, 2018

www.teledynedalsa.com

TELEDYNE DALSA
Everywhere**you**look™

NOTICE

Printed on September 12, 2018

Document Number: OC- COMM-GEVP0
Printed in Canada

About This Manual

This manual exists in Adobe Acrobat® (PDF) formats (printed manuals are available as special orders). The PDF format make full use of hypertext cross-references. The Teledyne DALSA home page on the Internet, located at http://www.teledynedalsa.com/imaging, contains documents, software updates, demos, errata, utilities, and more.

About Teledyne DALSA

Teledyne DALSA is an international high performance semiconductor and electronics company that designs, develops, manufactures, and markets digital imaging products and solutions, in addition to providing wafer foundry services.

Teledyne DALSA Digital Imaging offers the widest range of machine vision components in the world. From industry-leading image sensors through powerful and sophisticated cameras, frame grabbers, vision processors and software to easy-to-use vision appliances and custom vision modules.

Contents

GigE-V Framework for Linux Overview

A Compact API for GigE Vision Cameras under Linux

This document describes the GigE-V Framework for Linux which is a simplified, user-level API for accessing the features of GigE Vision devices. Its compact footprint is ideal for embedded platforms.

It is implemented in the C language and has an operating system independent layer that allows it to run, potentially, on any operating system which supports threads, events, and a socket based network interface. This implementation is intended Linux and is supported on most popular distributions such as Ubuntu, Debian, Suse/openSuse, and Red Hat (RHEL/Fedora/CentOS/Scientific).

Long-Term Support Linux Distributions

Teledyne DALSA recommends using long-term support (LTS) Linux distributions for machine vision applications. Long term support guarantees that critical system updates are available to ensure system stability.

Suitable long term support distibutions include:

- **Ubuntu**: Released every 2 years. Starting with Ubuntu 12.04 LTS, versions receive five years support.
- **CentOS**: Release distributions provide long term support (*end of life*) for up to 10 years. For example, CentOS-7 provides updates until June 30, 2024.
- **Red Hat**: Red Hat Enterprise Linux versions 6 and 7 each deliver 10 years of support (*life cycle*), unless otherwise noted, in Full Support, Maintenance Support 1 and Maintenance Support 2 Phases followed by an Extended Life Phase. In addition, for Red Hat Enterprise Linux 6, customers may purchase annual Add-on subscriptions called *Extended Life-cycle Support* (ELS) to extend limited subscription services beyond the Maintenance Support 2 Phase.

Supported PC and Embedded Hardware Platforms

The following PC architectures are supported:

- **x86** : Intel/AMD 32-bit and 64-bit CPUs

The following embedded architectures are supported:

- **ARM AArch64**: 64-bit ARMv8
- **ARM hard float** : 32-bit ARMv7 with hardware floating point
- **ARM soft float** : 32-bit ARM with software emulated floating point

System Requirements

- Linux OS support for Gigabit NIC hardware is required (kernel 2.6.24 and later)
- Support for PF_PACKET with RX_RING capability recommended for best performance (usually available with the Wireshark application and/or the libpcap package which is widely installed by default).
- **libcap-dev** package is required to use Linux “capabilities” when running as “root” is not desired.
- **libx11-dev** / **libxext-dev** packages are required for using the X11 display in the example programs.
- **libglade2-dev** package is required for building and using the GigE Vision Device Status tool (uses gtk).

See Appendix A: Common Package Management methods in Linux for information on installing the required packages and the various commands available.

ℹ	**Note:** It is recommended to enable “jumbo” frames by setting the NIC MTU to its maximum value (usually 9018). This can be set using “ifconfig” or a distribution-specific tool or configuration file. Please consult the documentation for the Linux distribution being used.

Application Notes

Available application notes for the GigE-V Framework are on the Teledyne DALSA website.

Getting Started

The GigE-V Framework for Linux is distributed as a compressed tar archive, with file type ".tar.gz". The naming convention e of this archive is:

GigE-V-Framework_<architecture>_<Version#>.<Build#>.tar.gz

For example, the 4 available files for version 2.02 build 0.0135 are:

- GigE-V-Framework_x86_2.02.0.0135.tar.gz,
- GigE-V-Framework_aarch64_2.02.0.0135.tar.gz,
- GigE-V-Framework_ARMhf_2.02.0.0135.tar.gz, and
- GigE-V-Framework_ARMsf_2.02.0.0135.tar.gz

At this time, only target systems configured for self-hosted development are supported. At installation time, parts of the API are compiled and linked to the run-time libraries found on the target system. This reduces the risk of an installation package failing to work with a target system due to mismatched versions of run-time libraries. As a consequence of this, certain pre-requisites are required for successful installation.

Prerequisites

To compile and link the API on installation and use the example applications that are distributed with the framework, installation of the following packages is required:

Package	Description
gcc	C compiler
g++	C++ compiler
GNU make	make utility
libX11-dev	Library for using basic X11 display primitives in programs
libxext-dev	Library for using extended X11 display primitives in programs
libgtk-3-dev	Compile and link GigE Vision Device Status tool
libglade2-0 libglade2-dev	Library for loading and using ".glade" UI definition files

In addition, the following libraries are useful for enhancing the performance of the framework.

Package	Description
libpcap0.8	Library for user level packet capture
libcap2	Library / tools for assigning Linux "capabilities" to a program
ethtool	Utility to configuring tuning parameters of NIC drivers (usually installed by default)

For example, in Ubuntu. packages can be installed from the terminal using the following command:

```
    sudo apt-get install <package name>
```

Note, if you are unable to locate a specific package, regular expression can be used to try to find a suitable alternative package. For example,

```
    sudo apt-get install libpcap*
```

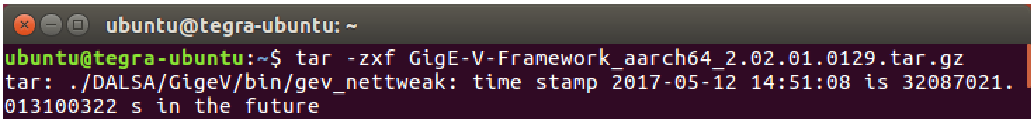
Note: The pre-requisite packages may have different names on different Linux distributions. See Appendix A: Common Package Management methods in Linux for more information on installing these packages and possible variations on their names.

System Date and Time Considerations

> **Note:** Some computer systems do not retain time and date settings after power cycling. This is particularly true of embedded systems. Installation of the GigE-V Framework for Linux can be affected by misconfigured time and date settings if the files being installed are timestamped in the future when compared to the current system time.

In such instances, it may be necessary to install/enable an NTP (Network Time Protocol) capability in order to keep the time and date settings current.

For example, the following message indicates the timestamp of the file is in the future:



```
ubuntu@tegra-ubuntu:~$ tar -zxf GigE-V-Framework_aarch64_2.02.01.0129.tar.gz
tar: ./DALSA/GigeV/bin/gev_nettweak: time stamp 2017-05-12 14:51:08 is 32087021.
013100322 s in the future
```

As an example, the ntpdate package can be installed and configured to use an available local or online NTP server to synchronize the system clock.

To install and configure the ntpdate package (using sample server address), use the following commands:

```
sudo apt-get install ntpdate
sudo ntpdate 140.165.161.1
```

It may be necessary to stop the service before initiating the update; for example:

```
sudo service ntp stop
sudo ntpdate time.nist.gov
sudo service ntp start
```

Additionally, the */etc/ntp.conf* file can be updated to include the required NTP server. For example, the following lines can be modified to add the NTP server (using sample server address):

```
# Use Ubuntu's ntp server as a fallback.
pool ntp.ubuntu.com
140.165.161.1
```

Installation

To install the GigE-V Framework for Linux from its compressed tar archive file, start by copying it to a base directory, usually the HOME directory of the user installing it, and extracting the files.

For example:

```
    cp GigE-V-Framework_x86_2.02.0.0132.tar.gz $HOME
    cd $HOME
    tar –zxf GigE-V-Framework_x86_2.02.0.0132.tar.gz
```

Then, change to the directory DALSA and run the installer script.

```
    cd DALSA
    ./corinstall
```

The script installs the GenICam SDK (v3_0 or later), if not already installed, and then configures, compiles, links, and installs the GigE-V Framework for Linux and its API libraries. It prompts for the administrator password when it needs to copy the various libraries to their preferred locations.

Alternately, the installation can be run using sudo (for example, using “sudo ./corinstall”).

The locations used for files are as follows:

Directory	Description
/opt/genicam_v3_0	GenICam SDK v3_0 files
/var/opt/genicam/xml/cache	GenICam XML cache
/usr/local/lib	Dynamic library files for the GigE-V Framework
/usr/dalsa/GigeV	Dynamic link to $HOME/DALSA/GigeV for system wide visibility

Environment Variables

The script also adds environment variables that are needed for the GenICam installation to operate properly. The environment variables added are :

```
    GENICAM_ROOT_V3_0 = /opt/genicam_v3_0
    GENICAM_CACHE_V3_0 = /var/opt/genicam/xml/cache
    GENICAM_LOG_CONFIG_V3_0 = /opt/genicam_v3_0/log/config-unix
```

and

```
    GIGEV_XML_DOWNLOAD = /usr/dalsa/GigeV
```

The new environment variables are visible to all subsequent login shells. After installation, for them to be visible, the current shell should be logged out and back in again. For the case of a GUI desktop, the user should log off and back in.

As a reminder, the installation script outputs the message :

```
    ************************************************************************************
    GenICam library installation was performed - you will need to log out and back in to
    properly set up the environment variables.
    *************************************************************************************
```

Note: The environment variables are set globally via shell scripts inside the folder `/etc/profile.d/` that are sourced at login. This configuration works for the shells *bash* and *csh* in most Linux systems.

Note: When using “sudo” to provide the necessary permissions for the higher performance interface, remember to use “sudo –E” or “sudo –i” to invoke an interactive (login) shell in order to pick up the environment variables that point to the GenICam SDK installation. These are used at runtime to be able to set up and use the GenICam XML based features.

Uninstalling

To uninstall the GigE-V Framework API, use the following steps:

```
cd $HOME/DALSA
./corinstall uninstall
```

The script prompts for the administrator password when deleting files from their install locations. In addition, the shell scripts that define the added environment variables are removed so that the environment variables will not be defined at the next login. Files unzipped from the *.tar* archive during installation are not removed.

Alternately, the uninstall procedure can be run using sudo (for example, using “sudo ./corinstall uninstall”).

The GenICam SDK, installed with the GigE-V Framework, is not uninstalled when this API is uninstalled since it may be used with other APIs and frameworks and with newer, updated, versions of this framework. During uninstallation, the following text reminds the user that GenICam is not uninstalled and describes how to uninstall the GenICam SDK if required:

```
********************************************************************************
   Found the GenICam library installation directory at /opt/genicam_v3_0
   It is not necessary to uninstall it if it will be re-used later

   To uninstall the GenICam library use the following command :

    . $HOME/DALSA/GenICam_v3_0_0_linux_pkg/uninstall.sh

(Then you will need to log out and log in to remove the environment variables)
********************************************************************************
```

```
Please note the command line for uninstall has a <space> between the <dot> and the
script name. As in <dot><space>$HOME/DALSA/GenICam_v3_0_0_linux_pkg /uninstall.sh
```

Performance Tuning

The Linux OS provides the GigE-V Framework with access to the standard network stack, suitable for grabbing single images, and also provides a high performance network packet access mechanism, suitable for streaming image sequences, that is traditionally used by packet sniffer applications.

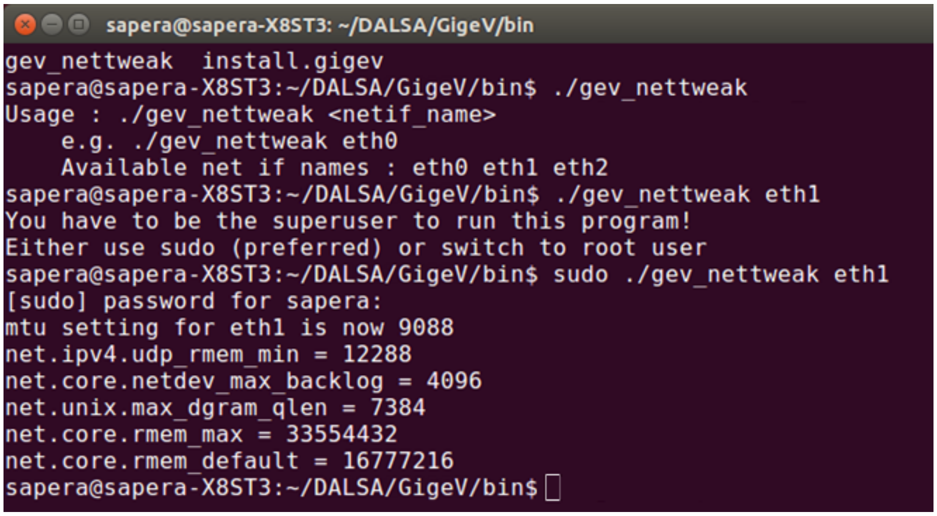
To avoid packet loss on the network interface, a number of parameters may be adjusted by the user. Important parameters to maximize are the MTU (maximum transmission unit) size and the number of receive buffers available to the NIC driver. This helps reduce the number of packets to process and therefore minimizes CPU overhead and interrupts.

A network tuning script provided with the API can maximize the MTU (enabling Jumbo frames) and optimize certain network settings, including the number of receive buffers, using a standard tool named "ethtool". The gev_nettweak tuning script is located in the following directory:

$HOME/DALSA/GigeV/bin/

For example, to adjust network interface eth0, use the following terminal command to run the script (administrator privileges are required):

```
sudo ./gev_nettweak eth0
```



```
gev_nettweak  install.gigev
sapera@sapera-X8ST3:~/DALSA/GigeV/bin$ ./gev_nettweak
Usage : ./gev_nettweak <netif_name>
    e.g. ./gev_nettweak eth0
    Available net if names : eth0 eth1 eth2
sapera@sapera-X8ST3:~/DALSA/GigeV/bin$ ./gev_nettweak eth1
You have to be the superuser to run this program!
Either use sudo (preferred) or switch to root user
sapera@sapera-X8ST3:~/DALSA/GigeV/bin$ sudo ./gev_nettweak eth1
[sudo] password for sapera:
mtu setting for eth1 is now 9088
net.ipv4.udp_rmem_min = 12288
net.core.netdev_max_backlog = 4096
net.unix.max_dgram_qlen = 7384
net.core.rmem_max = 33554432
net.core.rmem_default = 16777216
sapera@sapera-X8ST3:~/DALSA/GigeV/bin$
```

The “gev_nettweak” script adjusts the following parameters to assist the standard network stack in buffering more image data:

Parameter	Description
MTU	Maximizes the MTU (Maximal Tranmission Unit) size on the NIC. This corresponds to the maximum packet size for image data. The use of NIC hardware whose drivers support “Jumbo frames” aids in making this value as large as possible (typically maximum is around 9K bytes (9216 bytes).
net.ipv4.udp_rmem_min	Adjust the receive memory allocation size in the network stack.
net.core.netdev_max_backlog	Adjust the network packet backlog queue size.
net.unix.max_dgram_qlen	Adjust the network queue length for UDP packets. Computes the amount of memory for UDP packets - a maximum image size and the number of cameras expected provide a hint for this setting.
net.core.rmem_default net.core.rmem_max	Adjust the default (and maximum) memory for receiving network packets.
rx_value rx_jumbo	Use "ethtool" utility (if present) to adjust the setting of the network device drivers to optimize the rx_ring and the rx jumbo packet queue for maximum throughput and to disable the rx pause operation. This improves reception of image data packets from the cameras. (Sending to the camera is not as critical)

Access to the high performance packet access interface, mentioned above, is provided by the PF_PACKET socket interface and is restricted to processes that have a capability set that allows CAP_NET_RAW (permits raw access to an interface for capturing directly). Generally, this is accomplished either by using root / sudo permissions to run the program or to have the CAP_NET_RAW capability set up with the setcap utility that comes with the libcap library.

The ability to tune threads with specific CPU affinity values and higher priority is restricted to processes that have the capability set that allow CAP_SYS_NICE. Generally, this is accomplished either by using root / sudo permissions to run the program or to have the CAP_SYS_NICE capability set up with the setcap utility that comes with the libcap library. The CAP_SYS_NICE capability also allows the default scheduler (aka SCHED_OTHER) to be replaces with one of the real-time schedulers (SCHED_RR or SCHED_FIFO) for better thread handling performance.

Note: Some security environments can assign capabilities to executables with a configuration file (for example, /etc/permissions.local).

Without the CAP_NET_RAW bit set, the library defaults to standard packet accesses using sockets reading UDP (User Datagram Protocol) packets from the network stack. While the standard network socket access works for receiving images from a camera, there can be considerable latency in frame reception as the data makes its way through the network stack. For minimal latency and higher data rates, it is recommended that the PF_PACKET interface be used by enabling the CAP_NET_RAW capability bit.

Note: The setcap utility usage is “setcap cap_net_raw+eip <application>”. Where <application> is the file name of the executables being used. This includes the application program and all the loadable libraries it uses, referenced from ldconfig instead of LD_LIBRARY_PATH.

Note: When using “sudo” to provide the necessary permissions for the higher performance interface, remember to use “sudo –E” or “sudo –i” to invoke an interactive (login) shell in order to pick up the environment variables that point to the GenICam SDK installation. These are used at runtime to be able to set up and use the GenICam XML based features.

GigE Network Adapter Overview

GigE Vision compliant cameras connects to a computer’s Gigabit Network Adapter. If the computer is already connected to a network, the computer requires a second network adapter, either onboard or an additional PCIe NIC adapter.

IP Configuration Sequence

For Teledyne DALSA GigE Vision cameras IP (Internet Protocol) Configuration sequence to assign an IP address is executed automatically on camera power-up or when connected to a network. As a GigE Vision compliant device, the camera attempts to assign an IP address as follows.

For any GigE Vision device, the IP configuration protocol sequence is:

- Persistent IP (if enabled)
- DHCP (if a DHCP server is available)
- Link-Local Address (always enabled as default)

The factory default for Teledyne DALSA GigE Vision cameras is Persistent IP disabled and DHCP enabled with LLA always enabled as per the GigE Vision specification.

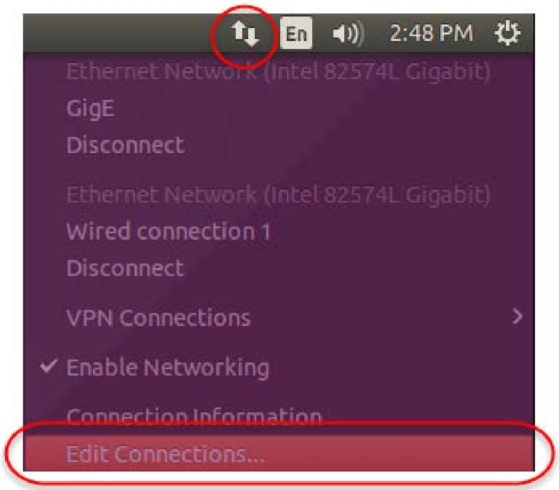
Supported Network Configurations

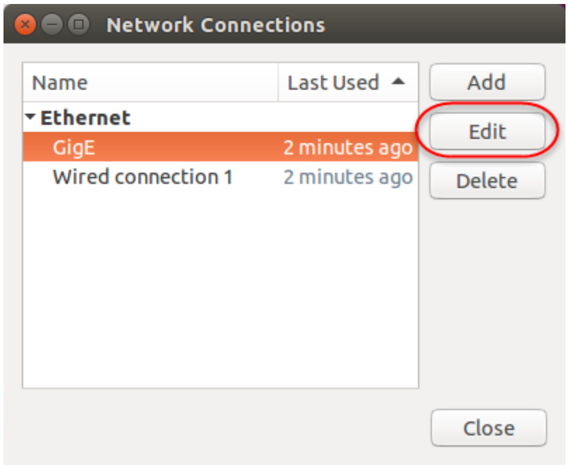
The camera obtains an IP address using the Link Local Address (LLA) or DHCP, by default. If required, a persistent IP address can be assigned (refer to the Network Imaging manual).

If a DHCP server is present on the network, the camera issues a DHCP request for an IP address. The DHCP server then provides the camera an IP address.

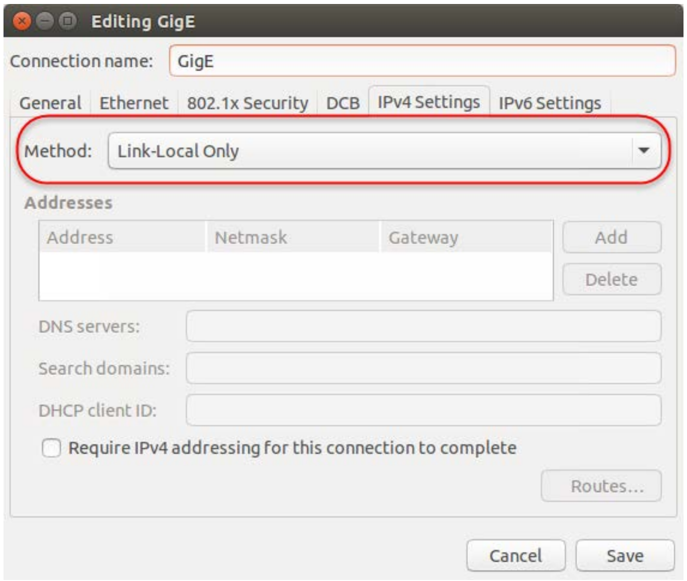
The LLA method, if used, automatically assigns the camera with a randomly chosen address on the 169.254.xxx.xxx subnet. After an address is chosen, the link-local process sends an ARP query with that IP onto the network to see if it is already in use. If there is no response, the IP is assigned to the device, otherwise another IP is selected, and the ARP is repeated. Note that the LLA mode is unable to forward packets across routers. To use LLA, the NIC must be configured to an address on the 169.254.xxx.xxx subnet.

For example, in Ubuntu, click the network icon in the menu bar to open the Network Connections dialog; select the NIC and click **Edit** to modify its parameters.





Name	Last Used
Ethernet	
GigE	2 minutes ago
Wired connection 1	2 minutes ago



Editing GigE

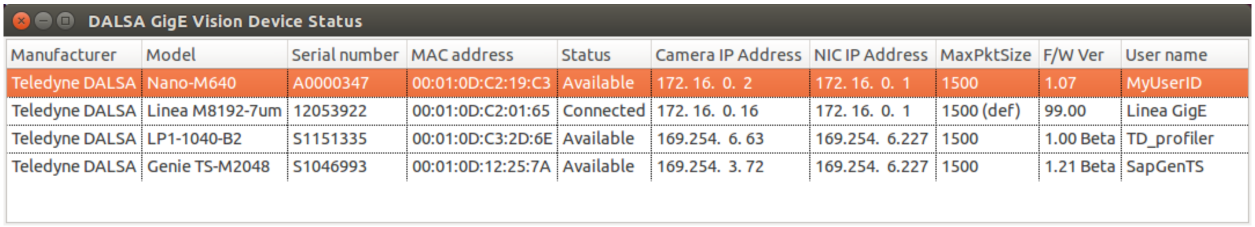
Connection name: GigE

Method: Link-Local Only

□ Require IPv4 addressing for this connection to complete

GigE Vision Device Status Tool

The GigE Vision Device Status tool lists all devices connected to the host system. Each GigE device is listed by name along with important information such as the assigned IP address and device MAC address.



DALSA GigE Vision Device Status

Manufacturer	Model	Serial number	MAC address	Status	Camera IP Address	NIC IP Address	MaxPktSize	F/W Ver	User name
Teledyne DALSA	Nano-M640	A0000347	00:01:0D:C2:19:C3	Available	172. 16. 0. 2	172. 16. 0. 1	1500	1.07	MyUserID
Teledyne DALSA	Linea M8192-7um	12053922	00:01:0D:C2:01:65	Connected	172. 16. 0. 16	172. 16. 0. 1	1500 (def)	99.00	Linea GigE
Teledyne DALSA	LP1-1040-B2	S1151335	00:01:0D:C3:2D:6E	Available	169.254. 6. 63	169.254. 6.227	1500	1.00 Beta	TD_profiler
Teledyne DALSA	Genie TS-M2048	S1046993	00:01:0D:12:25:7A	Available	169.254. 3. 72	169.254. 6.227	1500	1.21 Beta	SapGenTS

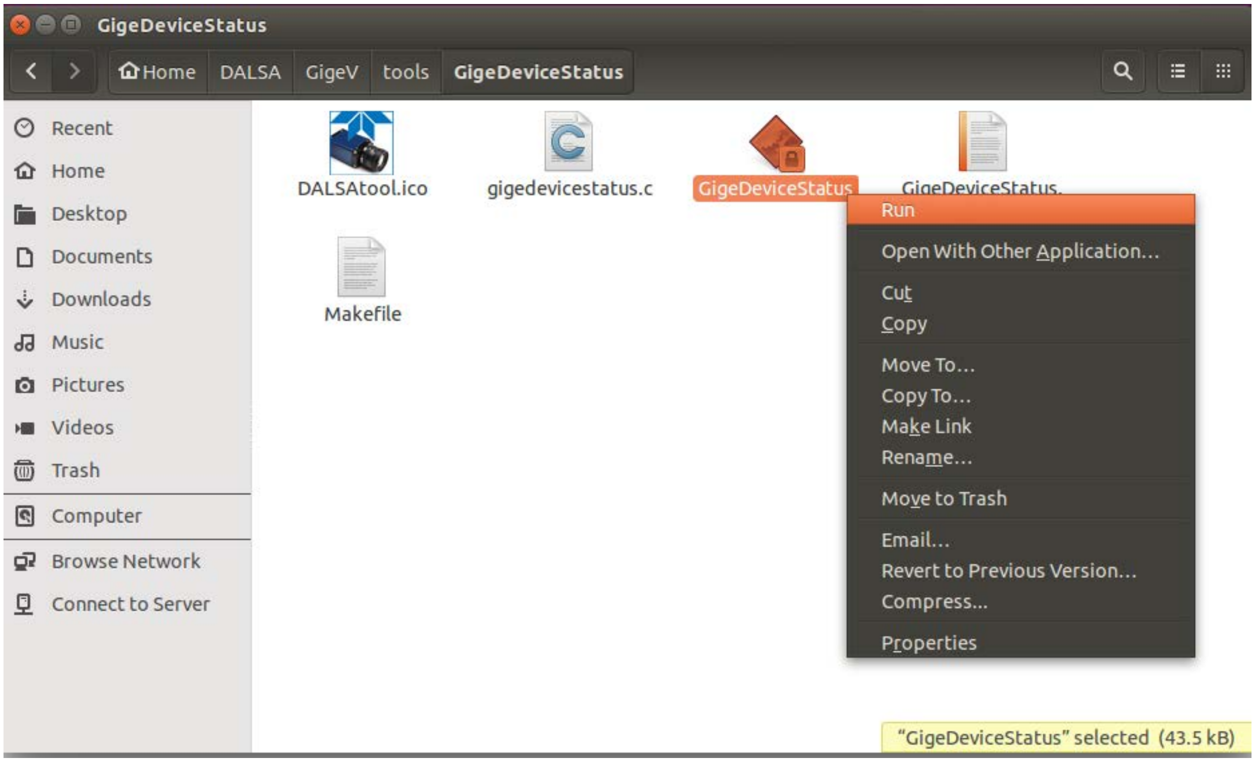
The following table provides the feature name and description of the available status fields.

Name	Feature Name	Description
Manufacturer	DeviceVendorName	Displays the device vendor name.
Model	DeviceModelName	Displays the device model name.
Serial number	DeviceSerialNumber	Displays the device’s factory set 8-digit serial number.
MAC address	deviceMacAddress	Displays the unique MAC (Media Access Control) address of the device.
Status	DeviceConnectionStatus	Displays the current status of the device connection. Possible values are: • Available: The device is available. • Connected: The device is currently connected to an application and is not available.
Camera IP Address	GevCurrentIPAddress	Displays the device’s current IP address.
NIC IP Address	GevPrimaryApplicationIPAddress	Displays the NIC IP address to which the device is connected.
MaxPktSize	GevSCPSPacketSize	Displays the current maximum packet size, in bytes, for the device to send on the stream channel. The actual packet size sent is set to the maximum supported by both the NIC and device packet size settings. Note, when a device is connected, this feature cannot be queried and displays a default value that may not

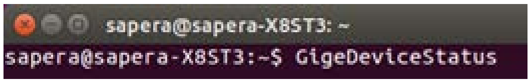
		correspond to the actual device setting.
F/W Version	DeviceVersion	Displays the device version. This field will also highlight if the firmware is a beta or custom design.
User name	DeviceUserID	Displays the device's current user-programmable identifier of up to 15 characters. The default factory setting is the camera serial number.

The GigE server periodically scans the network automatically to refresh its state. It might take a few seconds for the GigE Server to refresh its state after a GigE camera has obtained an IP address.

For example, to start the application in Ubuntu, use the File Manager to open the directory and use the pop-up menu Run command.

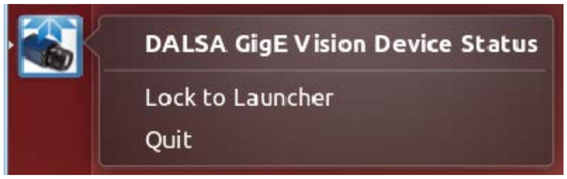


Alternatively, the tool can be started directly from any local directory (it is copied to the /usr/local/bin directory). For example, in Ubuntu:



```
sapera@sapera-X8ST3: ~
sapera@sapera-X8ST3:~$ GigeDeviceStatus
```

When the application is started, the application icon is placed in the Launcher bar (in Ubuntu), from where it can be locked to easily start the application.



lsgev Utility

The lsgev utility lists all GigE Vision devices connected to the host system using only the command line and a terminal. It can be used as an alternative to the GigE Vision Device Status tool for situations where using a GUI environment is not an option.

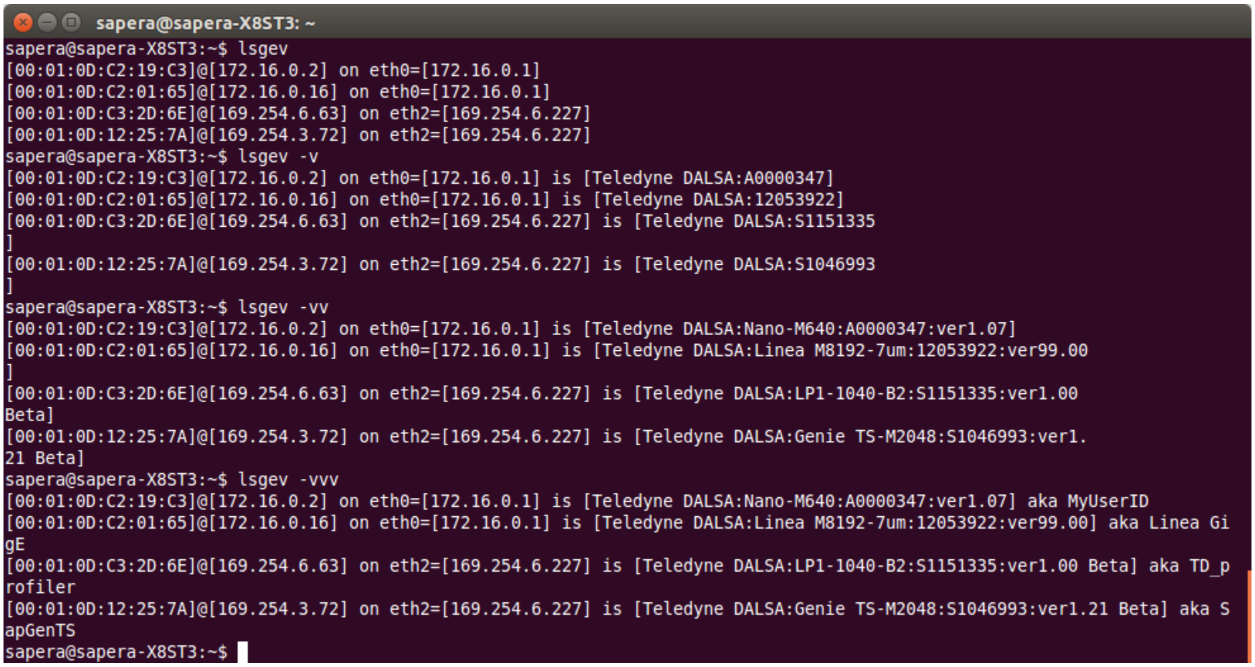
lsgev takes various command line options to alter the details reported for connected devices.

Usage: lsgev [options]

[options]

<none>	Output the MAC Address, IP Address, Net Name and NIC IP Address
-v	Verbose output: adds the Device Manufacturer and serial number string to the output
-vv	Very verbose output:adds the Model name and Version information to the output
-vvv	Most verbose output: adds the DeviceUserName string to the output
-?	Displays help (a list of the options with the format of their output) lsgev lists : <MAC>@[<CamIP>] on <netname>=[<NICIP>] lsgev -v lists : <MAC>@[<CamIP>] on <netname>=[<NICIP>] is <Manuf>:<Sn> lsgev -vv lists : <MAC>@[<CamIP>] on <netname>=[<NICIP>] is <Manuf>:<Model>:<Sn>:<Version> lsgev -vvv lists : <MAC>@[<CamIP>] on <netname>=[<NICIP>] is <Manuf>:<Model>:<Sn>:<Version> aka <UserName>

The following terminal output shows the different lsgev command line options with multilple NICs and cameras.

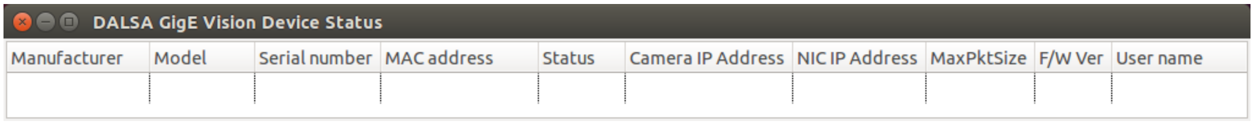


```
sapera@sapera-X8ST3:~$ lsgev
[00:01:0D:C2:19:C3]@[172.16.0.2] on eth0=[172.16.0.1]
[00:01:0D:C2:01:65]@[172.16.0.16] on eth0=[172.16.0.1]
[00:01:0D:C3:2D:6E]@[169.254.6.63] on eth2=[169.254.6.227]
[00:01:0D:12:25:7A]@[169.254.3.72] on eth2=[169.254.6.227]
sapera@sapera-X8ST3:~$ lsgev -v
[00:01:0D:C2:19:C3]@[172.16.0.2] on eth0=[172.16.0.1] is [Teledyne DALSA:A0000347]
[00:01:0D:C2:01:65]@[172.16.0.16] on eth0=[172.16.0.1] is [Teledyne DALSA:12053922]
[00:01:0D:C3:2D:6E]@[169.254.6.63] on eth2=[169.254.6.227] is [Teledyne DALSA:S1151335
]
[00:01:0D:12:25:7A]@[169.254.3.72] on eth2=[169.254.6.227] is [Teledyne DALSA:S1046993
]
sapera@sapera-X8ST3:~$ lsgev -vv
[00:01:0D:C2:19:C3]@[172.16.0.2] on eth0=[172.16.0.1] is [Teledyne DALSA:Nano-M640:A0000347:ver1.07]
[00:01:0D:C2:01:65]@[172.16.0.16] on eth0=[172.16.0.1] is [Teledyne DALSA:Linea M8192-7um:12053922:ver99.00
]
[00:01:0D:C3:2D:6E]@[169.254.6.63] on eth2=[169.254.6.227] is [Teledyne DALSA:LP1-1040-B2:S1151335:ver1.00
Beta]
[00:01:0D:12:25:7A]@[169.254.3.72] on eth2=[169.254.6.227] is [Teledyne DALSA:Genie TS-M2048:S1046993:ver1.
21 Beta]
sapera@sapera-X8ST3:~$ lsgev -vvv
[00:01:0D:C2:19:C3]@[172.16.0.2] on eth0=[172.16.0.1] is [Teledyne DALSA:Nano-M640:A0000347:ver1.07] aka MyUserID
[00:01:0D:C2:01:65]@[172.16.0.16] on eth0=[172.16.0.1] is [Teledyne DALSA:Linea M8192-7um:12053922:ver99.00] aka Linea Gi
gE
[00:01:0D:C3:2D:6E]@[169.254.6.63] on eth2=[169.254.6.227] is [Teledyne DALSA:LP1-1040-B2:S1151335:ver1.00 Beta] aka TD_p
rofiler
[00:01:0D:12:25:7A]@[169.254.3.72] on eth2=[169.254.6.227] is [Teledyne DALSA:Genie TS-M2048:S1046993:ver1.21 Beta] aka S
apGenTS
sapera@sapera-X8ST3:~$ 
```

Camera IP Address Configuration Tool

The gevipconfig tool is a command line utility that assigns an IP address to a camera based on its MAC address. IP addresses can be assigned temporarily (ForceIP) or with a persistent IP mode (assigned address is saved in non-volatile memory and used on power-up).

This allows cameras to be recovered if the network addressing scheme makes them undetectable. The gevipconfig tool can be used, for example, when the GigE Vision Device Status tool does not display any devices (with a camera properly powered and connected):



DALSA GigE Vision Device Status

Manufacturer	Model	Serial number	MAC address	Status	Camera IP Address	NIC IP Address	MaxPktSize	F/W Ver	User name

The command parameters are:

```
    Usage: gevipconfig [-p] MAC_Address IP_Address Subnet_Mask
                        -p (optional) = sets address/subnet to persistent mode
                        MAC_Address   = aa:bb:cc:dd:ee:ff (a-f are HEX digits)
                        IP_Address    = A.B.C.D  (A-D are decimal digits)
                        Subnet_Mask   = A.B.x.y  (Mask for class B or C subnet)
```

The tool can be started directly from the local directory (it is copied to the /usr/local/bin directory).

Example usage:

To temporarily set a camera with MAC address 00:01:0D:11:08:7F to an address visible to a NIC (for example, in LAA mode IP address 169.254.0.1 with subnet 255.255.0.0):

```
gevipconfig 00:01:0D:11:08:7F 169.254.8.128 255.255.0.0
```

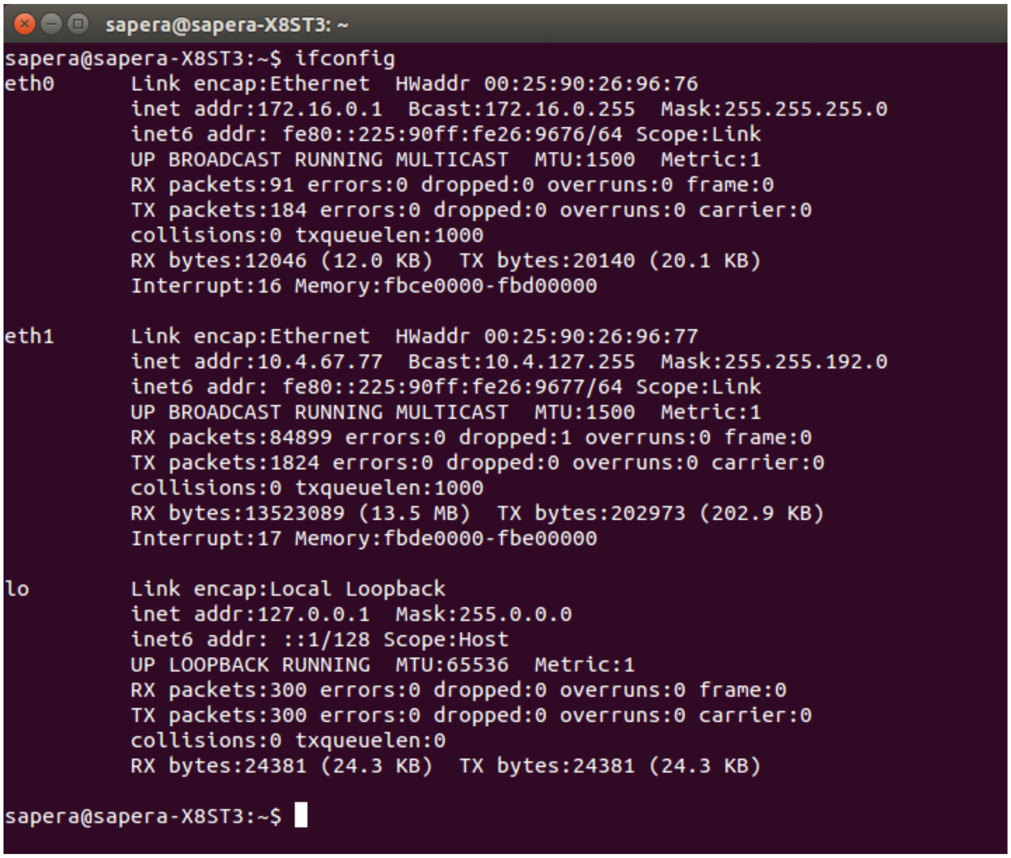
Note: The camera will retain its previous settings when reset.

Setting A Persistent IP Address

To set a camera with MAC address 00:01:0D:11:08:7F to a persistent static address of 172.10.1.4 (camera reboots with the specified address):

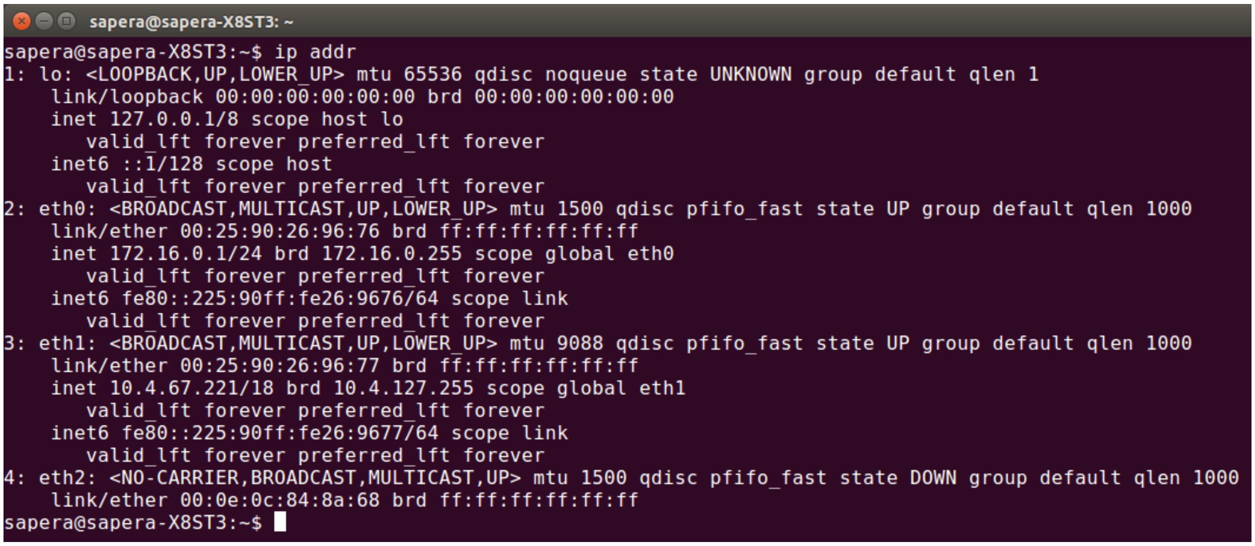
```
gevipconfig –p 00:01:0D:11:08:7F 172.10.1.4 255.255.255.0
```

The `ifconfig` command can be used to list the available NIC IP configurations.



```
sapera@sapera-X8ST3:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:25:90:26:96:76
          inet addr:172.16.0.1  Bcast:172.16.0.255  Mask:255.255.255.0
          inet6 addr: fe80::225:90ff:fe26:9676/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:91 errors:0 dropped:0 overruns:0 frame:0
          TX packets:184 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:12046 (12.0 KB)  TX bytes:20140 (20.1 KB)
          Interrupt:16 Memory:fbce0000-fbd00000

eth1      Link encap:Ethernet  HWaddr 00:25:90:26:96:77
          inet addr:10.4.67.77  Bcast:10.4.127.255  Mask:255.255.192.0
          inet6 addr: fe80::225:90ff:fe26:9677/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:84899 errors:0 dropped:1 overruns:0 frame:0
          TX packets:1824 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13523089 (13.5 MB)  TX bytes:202973 (202.9 KB)
          Interrupt:17 Memory:fbde0000-fbe00000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:300 errors:0 dropped:0 overruns:0 frame:0
          TX packets:300 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:24381 (24.3 KB)  TX bytes:24381 (24.3 KB)

sapera@sapera-X8ST3:~$ 
```

Alternately, the ip command can be used, with the “addr” switch, to show available network links.



```
sapera@sapera-X8ST3:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:25:90:26:96:76 brd ff:ff:ff:ff:ff:ff
    inet 172.16.0.1/24 brd 172.16.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::225:90ff:fe26:9676/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9088 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:25:90:26:96:77 brd ff:ff:ff:ff:ff:ff
    inet 10.4.67.221/18 brd 10.4.127.255 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::225:90ff:fe26:9677/64 scope link
       valid_lft forever preferred_lft forever
4: eth2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN group default qlen 1000
    link/ether 00:0e:0c:84:8a:68 brd ff:ff:ff:ff:ff:ff
sapera@sapera-X8ST3:~$ 
```

GigE with TurboDrive

The GigE-V Framework supports devices equipped with TurboDrive™ technology, delivering high speed data transfers exceeding the GigE limit. TurboDrive uses advanced data modeling to boost data transfers up to 2 or 3 times faster than standard GigE Vision speeds – with no loss of image quality. These breakthrough rates are achieved using a proprietary, patent pending process that assembles data from the sensor to optimize throughput, simultaneously taking full advantage of both the sensor's maximum frame rate and the camera's maximum GigE data transfer speed (up to 115 Mbytes/s). Teledyne DALSA's TurboDrive increases system dependability and robustness similar to Camera Link throughput on a GigE network.

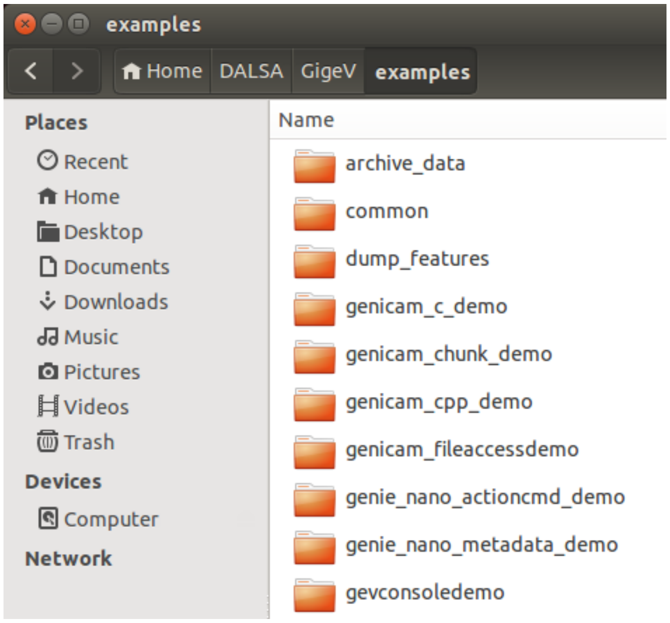
The "*transferTurboMode*" feature sets the enable state of TurboDrive (1 = enable, 0 = disable). If TurboDrive is not supported this feature returns an error. Refer to the example programs (genicam_c_demo/genicam_cpp_demo) for source code on to how to utilize TurboDrive in your application.

Important: Actual Transfers with TurboDrive are image content dependent but in the best case scenario, transfers over a GigE Network can reach the camera's internal acquisition limit of up to 252MB/sec. If transfers are less than the camera maximum acquisition rate, camera memory will be used as a circular frame buffer. Refer to the TurboDrive Primer on the Teledyne DALSA web site for more details.

Example Programs

Example programs are located in the following directory:

$HOME/DALSA/GigeV/examples



archive_data
common
dump_features
genicam_c_demo
genicam_chunk_demo
genicam_cpp_demo
genicam_fileaccessdemo
genie_nano_actioncmd_demo
genie_nano_metadata_demo
gevconsoledemo

The example programs are categorized by the basic functionality they demonstrate.

Grab / Display Frame Data	Description
genicam_c_demo	The genicam_c_demo program demonstrates a grab and display application using only C language calls to the Framework API.
genicam_cpp_demo	The genicam_cpp_demo program demonstrates a grab and display application along with access to the underlying GenICam GenApi classes.
genicam_chunk_demo	The genicam_chunk_demo program demonstrates how to extract metadata from a frame buffer using the chunk data layout information provided by the GenICam XML file. Grab and display functionality is also demonstrated.
genie_nano_metadata_demo	The genie_nano_metadata program demonstrates how to access metadata in a frame by directly accessing the memory location in the frame buffer containing the fixed chunk data layout provided by the Genie Nano family of cameras. Grab and display functionality is also demonstrated.
gevconsoledemo	This demo is provided for compatibility with older DALSA cameras; use newer demos for reference when creating applications. The gevconsoledemo program demonstrates a grab and display application utilizing direct register access to the camera. Only cameras known to the API can be used with this program since the camera register definitions need to be hardcoded in a static table. For more information, please see Appendix C: Feature Access Through Static Registers.
Archive Frame Data	**Description**
save_data_demo	The save_data_demo (in $HOME/DALSA/GigeV/examples/archive_data) saves frame data to a container file (of type ".gevbuf"). It saves single frames or sequences of frames to the ".gevbuf" containers with options to include metadata (chunk data), and to control the type of image data decoding performed prior to storage.
restore_nano_data_demo	The restore_nano_data_demo (in $HOME/DALSA/GigeV/examples/archive_data) restores frame data from the variable length ".gevbuf" containers created by the save_data_demo. The program demonstrates how to recover the frame data, perform any decoding required to generate usable images, and optionally save the image data to

	TIFF files. If metadata (chunk data) is present in the container, it is accessed as the fixed chunk data layout provided by the Genie Nano family of cameras.
GenICam Feature Access	**Description**
dumpfeatures	The dumpfeatures program (in $HOME/DALSA/GigeV/examples/dump_features) demonstrates how to access the GenICam XML features of a camera and output the entire hierarchy of features, including their type, to the screen.
savefeatures	The savefeatures program (in $HOME/DALSA/GigeV/examples/dump_features) demonstrates how to save the streamable features, as {feature_name : value} pairs, to the screen or to a text file.
loadfeatures	The loadfeatures program (in $HOME/DALSA/GigeV/examples/dump_features) demonstrates how to load {feature_name : value} pairs, from a text file, to the camera.
c_loadfeatures	The c_loadfeatures program (in $HOME/DALSA/GigeV/examples/dump_features) demonstrates loading {feature_name : value } pairs from a text file to the camera using only C callable functions from the Framework API.
genicam_fileaccessdemo	The genicam_fileaccessdemo program demonstrates access to the file interface on the camera. The files present can be detected and read or written, as allowed by the definitions provided by the GenICam XML file associated with the camera.
GigE Vision ACTION_CMD	**Description**
nano_trigger_demo	The nano_trigger_demo (in $HOME/DALSA/GigeV/examples/genie_nano_actioncmd_demo) demonstrated how to set up a camera for use with the action_cmd_demo. The demo sets camera features to acquire images from a trigger provided by an action command (Action1) using the default ACTION_CMD functionality provided by the Genie Nano family of cameras.
action_cmd_demo	The action_cmd_demo (in $HOME/DALSA/GigeV/examples/genie_nano_actioncmd_demo) demonstrates how to send an action command to cameras on a network. The demo sends an ACTION_CMD specific to the Genie Nano family of cameras to be identified as Action1. The accompanying nano_trigger_demo will receive frames triggered by Action1.

Note: Running demos that display images, such as genicam_c_demo and genicam_cpp_demo, on an ARM hard float platform using the ARM soft float package (GigE-V-Framework_ARMsf_xxx) will not execute properly unless the required soft float library packages are installed.

Each example program directory includes a makefile to compile the example. Examples must be compiled before using by running the make command in the example directory. For example, in Ubuntu:

```
sapera@computername:~/DALSA/GigeV/examples/genicam_c_demo$ make
```

Note: If the make operation fails on link, verify that the required prerequisites are installed for the given hardware architecture (for example, ARM hardfloat, ARM softfloat, and Intel x86).

Call the program name to run program. For example, in Ubuntu, to run the program in the current directory, precede the program name with “./”:

```
sapera@computername:~/DALSA/GigeV/examples/genicam_c_demo$ ./genicam_c_demo
```

If multiple cameras are connected, most example programs can be invoked using a camera index (starting from 0):

```
./genicam_c_demo 1
```

Note: For multiple cameras on the same NIC indices are not static and are populated dynamically when the program is run, therefore the index for a specific camera may change depending on the order it is acknowledged when the program is run. Functions are provided to perform automatic camera (device) discovery and enumeration; see the Automatic Camera Discovery section. Functions are also available to open cameras by IP address, name or serial number; see the Connecting to a Camera section for more information.

The "-" or "?" switch provides usage for most example programs. For example,

```
ubuntu@tegra-ubuntu:~/DALSA/GigeV/examples/dump_features$ ./savefeatures ?

GigE Vision Library GenICam Feature Save Example (May  6 2016)
3 camera(s) on the network
Usage: savefeatures                     : Output features from camera 0 to stdout.
       savefeatures - cam_index         : Output features from camera 'cam_index' to
stdout. (Note the hyphen indicating stdout)
       savefeatures filename            : Save features from camera 0 to 'filename'.
       savefeatures filename cam_index : Save features from camera 'cam_index' to
'filename'.
ubuntu@tegra-ubuntu:~/DALSA/GigeV/examples/dump_features$
```

Image Display

The example programs include code for a rudimentary image display capability using function calls to the low level X11 libraries. These functions provide access to the X server underlying most, if not all, graphical environments available in Linux. All of the source code showing how to prepare and display an image buffer is provided.

Note: The display functions are not optimized for speed or efficiency and are intended as a simple mechanism for rendering an image buffer to the screen for viewing.

Saving Images

Example programs that support image acquisition (excluding the gevconsoledemo) allow saving images as TIFF files using the "@" command, which saves the last acquired image (image data only; no metadata is included).

Both the grab demos (genicam_c_demo and genicam_cpp_demo), as well as the archive data examples (save_data_demo and restore_nano_data_demo) and genie_nano_metadata_demo, support saving images as TIFF files.

Bayer Conversion

By default, example programs that support saving frames enable conversion of Bayer to RGB format.

If ENABLE_BAYER_CONVERSION is 1 (default), Bayer formats display as color and are stored to TIFF as color.

If ENABLE_BAYER_CONVERSION is 0 (edit and recompile), Bayer formats display as mono and are stored to TIFF as mono.

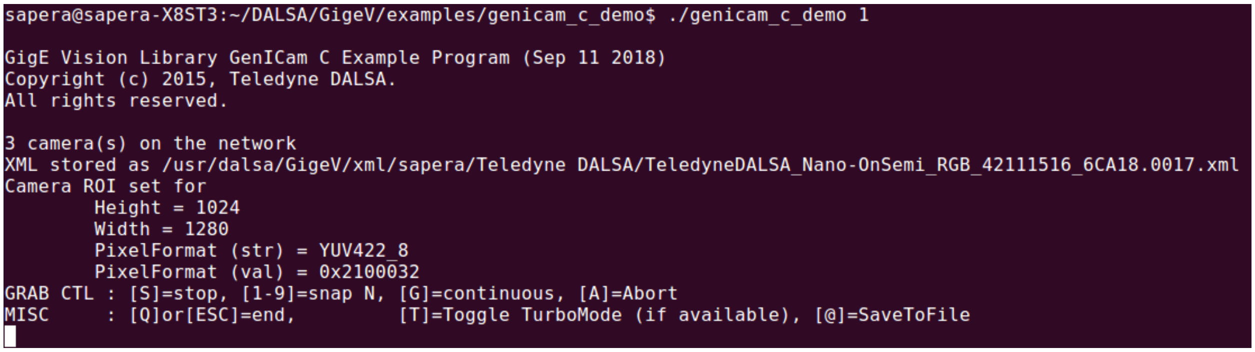
Grab Demos

The grab demo examples (genicam_c_demo and genicam_cpp_demo) demonstrate how to acquire and display images using a continuous (grab) or single frame (snap) acquisition. TurboDrive The examples display the current image and pixel format settings for the selected camera. Bayer/YUV images are displayed as monochrome since no conversion is performed; RGB images are displayed in color.

For a list of supported pixel formats refer to the Supported Pixel Formats section.

When more than 1 camera is available on the network, call the executable followed by the camera index (0 to (number of cameras -1), default = 0). For example,

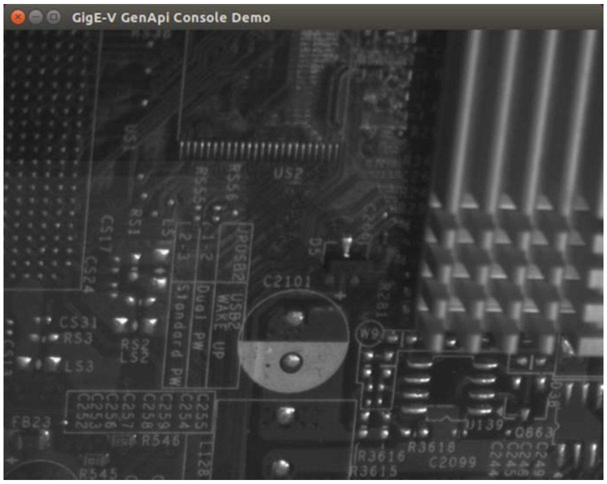
```
./genicam_c_demo 1
```



```
sapera@sapera-X8ST3:~/DALSA/GigeV/examples/genicam_c_demo$ ./genicam_c_demo 1

GigE Vision Library GenICam C Example Program (Sep 11 2018)
Copyright (c) 2015, Teledyne DALSA.
All rights reserved.

3 camera(s) on the network
XML stored as /usr/dalsa/GigeV/xml/sapera/Teledyne DALSA/TeledyneDALSA_Nano-OnSemi_RGB_42111516_6CA18.0017.xml
Camera ROI set for
        Height = 1024
        Width = 1280
        PixelFormat (str) = YUV422_8
        PixelFormat (val) = 0x2100032
GRAB CTL : [S]=stop, [1-9]=snap N, [G]=continuous, [A]=Abort
MISC     : [Q]or[ESC]=end,         [T]=Toggle TurboMode (if available), [@]=SaveToFile
```

The image is displayed in a separate window . To improve display performance, the user can optimize the display as needed for the required platform.

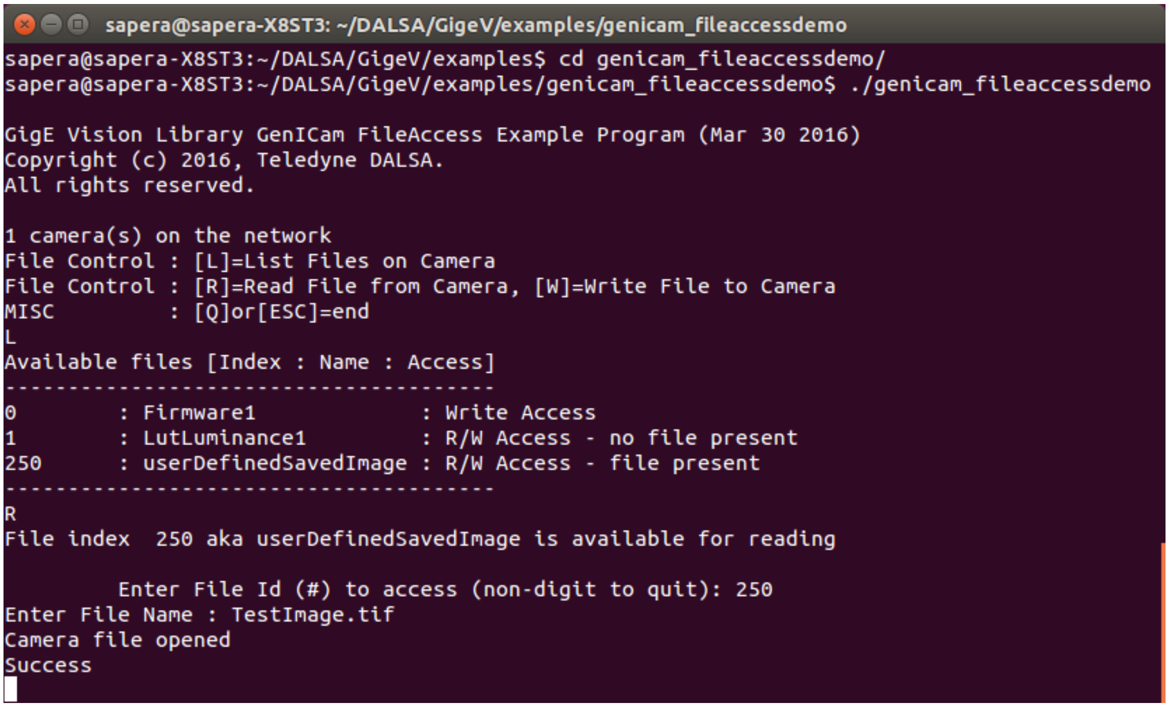
Note: Depending on the image size, the display window can overlap the terminal window; switch focus to the terminal window as required.



File Access Example

The file access example provides commands to list (L) the available files and their associated file access privileges, read (R) files to save in the current directory, and write (W) files from the current directory to the camera. Indices identify the available files.

Note: Refer to the camera documentation for the available files, formats and usage.


sapera@sapera-X8ST3: ~/DALSA/GigeV/examples/genicam_fileaccessdemo

```
sapera@sapera-X8ST3:~/DALSA/GigeV/examples$ cd genicam_fileaccessdemo/
sapera@sapera-X8ST3:~/DALSA/GigeV/examples/genicam_fileaccessdemo$ ./genicam_fileaccessdemo

GigE Vision Library GenICam FileAccess Example Program (Mar 30 2016)
Copyright (c) 2016, Teledyne DALSA.
All rights reserved.

1 camera(s) on the network
File Control : [L]=List Files on Camera
File Control : [R]=Read File from Camera, [W]=Write File to Camera
MISC         : [Q]or[ESC]=end
L
Available files [Index : Name : Access]
---------------------------------------
0        : Firmware1             : Write Access
1        : LutLuminance1         : R/W Access - no file present
250      : userDefinedSavedImage : R/W Access - file present
---------------------------------------
R
File index  250 aka userDefinedSavedImage is available for reading

         Enter File Id (#) to access (non-digit to quit): 250
Enter File Name : TestImage.tif
Camera file opened
Success
```

Feature Access Examples

Feature access examples include the *dumpfeatures*, *savefeatures* and *loadfeatures/c_loadfeatures* that demonstrate how to list the available features on a camera, output the current camera settings and load camera settings to the camera, respectively.

The *dumpfeatures* example parses the xml file to extract all available features on the camera by category and their corresponding type, displaying them in the terminal window:

```
Dumping feature tree :
    Category : Root
             Category : deviceInformation
                      DeviceVendorName : <IString>
                      DeviceFamilyName : <IString>
                      DeviceModelName : <IString>
                      DeviceVersion : <IString>
                      deviceManufacturerPartNumber : <IString>
                      DeviceManufacturerInfo : <IString>
                      DeviceFirmwareVersion : <IString>
                      DeviceID : <IString>
                      DeviceSerialNumber : <IString>
                      deviceMacAddress : <IInteger>
…
             Category : deviceSensorControl
                      DeviceScanType : <IEnumeration>
                      sensorColorType : <IEnumeration>
                      pixelSizeInput : <IEnumeration>
                      SensorWidth : <IInteger>
```

```
                        SensorHeight : <IInteger>
                        acquisitionFrameRateControlMode : <IEnumeration>
                        AcquisitionFrameRateEnable : <IBoolean>
                        AcquisitionFrameRate : <IFloat>
…
                Category : DigitalIOControl
                        TriggerSelector : <IEnumeration>
                        TriggerMode : <IEnumeration>
                        triggerFrameCount : <IInteger>
…
```

The *savefeatures* and *loadfeatures/c_loadfeatures* examples export/import feature settings that are streamable (that is, can be uploaded/downloaded in a batch process) using a simple text file in the following format:

```
<feature> <value>
<feature> <value>
```

For example, to save current camera feature settings to a text file (in the current directory), use the following command:

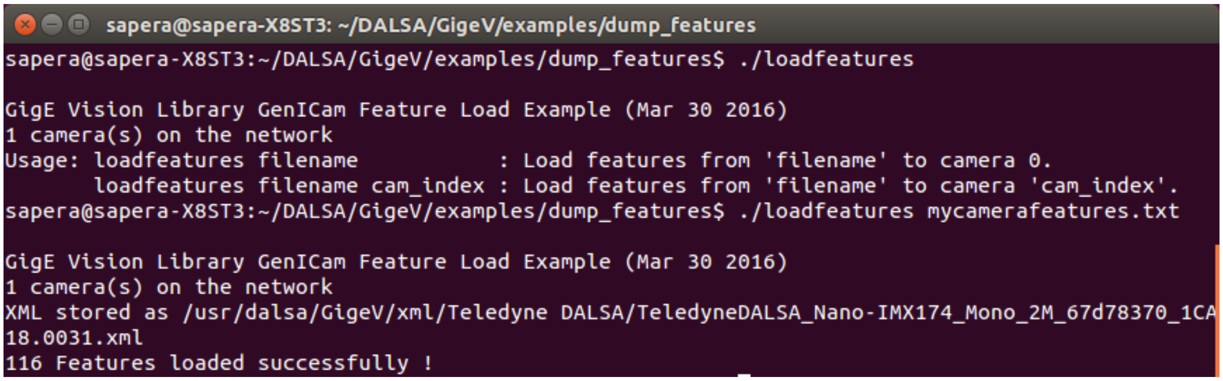
```
    ./savefeatures <filename>.txt
```

With multiple cameras, usage is as follows:

```
    savefeatures                    : Output features from camera 0 to stdout.
    savefeatures - cam_index        : Output features from camera 'cam_index' to
    stdout. (Note the hyphen indicating stdout)
    savefeatures filename           : Save features from camera 0 to 'filename'.
    savefeatures filename cam_index : Save features from camera 'cam_index' to
    'filename'.
```

When loading features, the file need only contain the feature-value pair for those features that need to be modified. For example:

```
    PixelFormat Mono8
    OffsetX 0
    OffsetY 0
    Width 640
    Height 480
```



```
sapera@sapera-X8ST3: ~/DALSA/GigeV/examples/dump_features
sapera@sapera-X8ST3:~/DALSA/GigeV/examples/dump_features$ ./loadfeatures

GigE Vision Library GenICam Feature Load Example (Mar 30 2016)
1 camera(s) on the network
Usage: loadfeatures filename           : Load features from 'filename' to camera 0.
       loadfeatures filename cam_index : Load features from 'filename' to camera 'cam_index'.
sapera@sapera-X8ST3:~/DALSA/GigeV/examples/dump_features$ ./loadfeatures mycamerafeatures.txt

GigE Vision Library GenICam Feature Load Example (Mar 30 2016)
1 camera(s) on the network
XML stored as /usr/dalsa/GigeV/xml/Teledyne DALSA/TeledyneDALSA_Nano-IMX174_Mono_2M_67d78370_1CA
18.0031.xml
116 Features loaded successfully !
```

If multiple cameras are connected, the camera index (0 to (number of cameras -1), default = 0) is used to select the required camera.

Note: Not all camera features are streamable; for non-streamable features you must use the GevGetFeatureValue and GevSetFeatureValue functions.

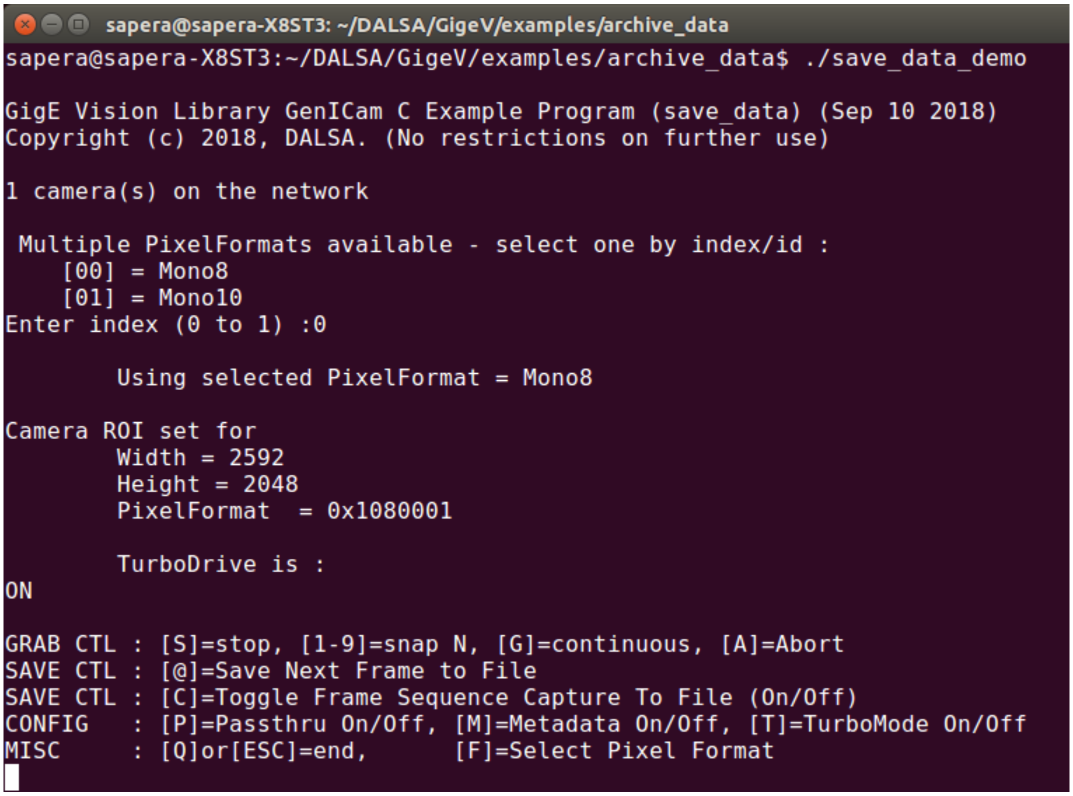
Archive Data Examples

The archive data examples include the save_data_demo and restore_nano_data_demo programs. These programs use a ".gevbuf" file format to store frames or frame sequences (with or without the metadata appended to each frame). The ".gevbuf" file is a variable length container for frame payloads / sequences of frame payloads, intended for archiving frames in raw, unprocessed form.

All functions for manipulating gevbuf files are included in the demo source code.

save_data_demo

The save_data_demo is intended for archiving frames. It provides a "passthrough" option that disables all processing in the acquisition mechanism; that is, packed data remains packed and TurboDrive encoded frames remained compressed. The program can also enable/disable TurboDrive compression and metadata.

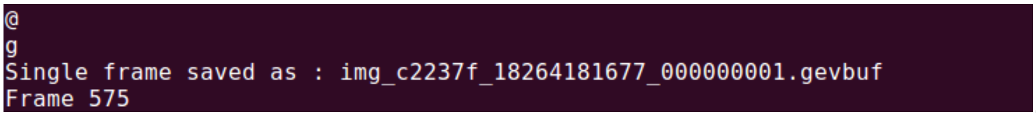


```
sapera@sapera-X8ST3:~/DALSA/GigeV/examples/archive_data$ ./save_data_demo

GigE Vision Library GenICam C Example Program (save_data) (Sep 10 2018)
Copyright (c) 2018, DALSA. (No restrictions on further use)

1 camera(s) on the network

 Multiple PixelFormats available - select one by index/id :
    [00] = Mono8
    [01] = Mono10
Enter index (0 to 1) :0

        Using selected PixelFormat = Mono8

Camera ROI set for
        Width = 2592
        Height = 2048
        PixelFormat  = 0x1080001

        TurboDrive is :
ON

GRAB CTL : [S]=stop, [1-9]=snap N, [G]=continuous, [A]=Abort
SAVE CTL : [@]=Save Next Frame to File
SAVE CTL : [C]=Toggle Frame Sequence Capture To File (On/Off)
CONFIG   : [P]=Passthru On/Off, [M]=Metadata On/Off, [T]=TurboMode On/Off
MISC     : [Q]or[ESC]=end,      [F]=Select Pixel Format
```

When acquiring frames, the frame number is displayed in the console window.

The "@" command saves the next frame acquired to file, therefore this command should be followed by a snap [1-9] or continuous grab command [G] (when the grab is stopped using the [S] command, the first frame of the grab is saved).

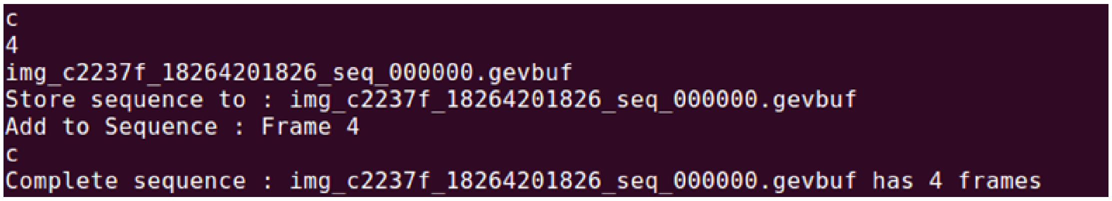
The program saves frames and sequences using the custom ".gevbuf" format in the archive_data example directory; the filename is displayed in the console:



```
@
g
Single frame saved as : img_c2237f_18264181677_000000001.gevbuf
Frame 575
```

.

To save a sequence of frames, use the "C" command to start and stop the sequence capture. When a capture is started, frames will continue to be added (while starting/stopping acquisition) to the sequence until the capture is stopped.

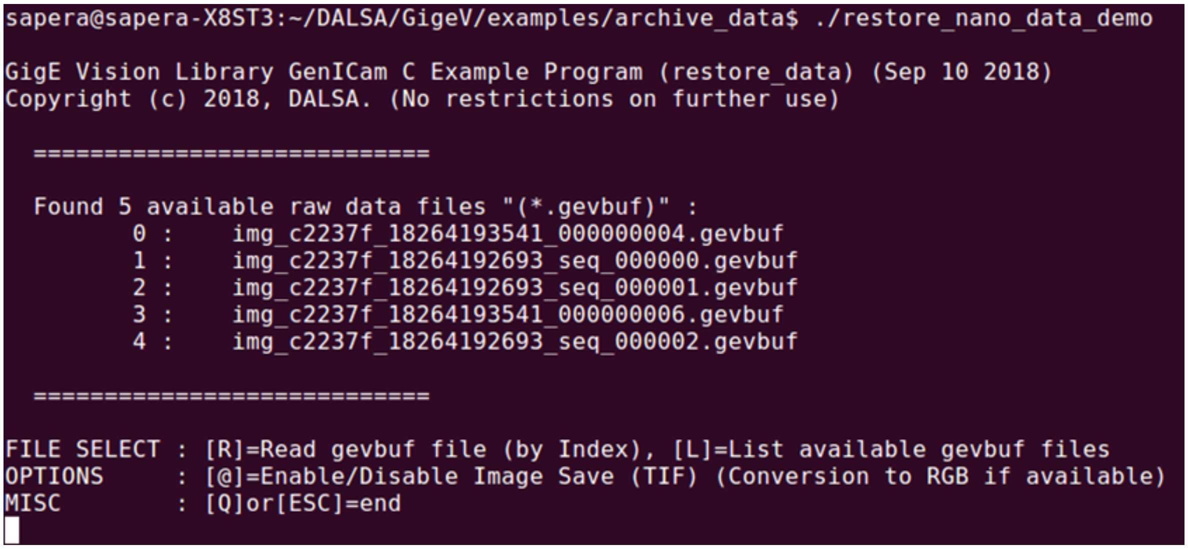


```
c
4
img_c2237f_18264201826_seq_000000.gevbuf
Store sequence to : img_c2237f_18264201826_seq_000000.gevbuf
Add to Sequence : Frame 4
c
Complete sequence : img_c2237f_18264201826_seq_000000.gevbuf has 4 frames
```

restore_nano_data_demo

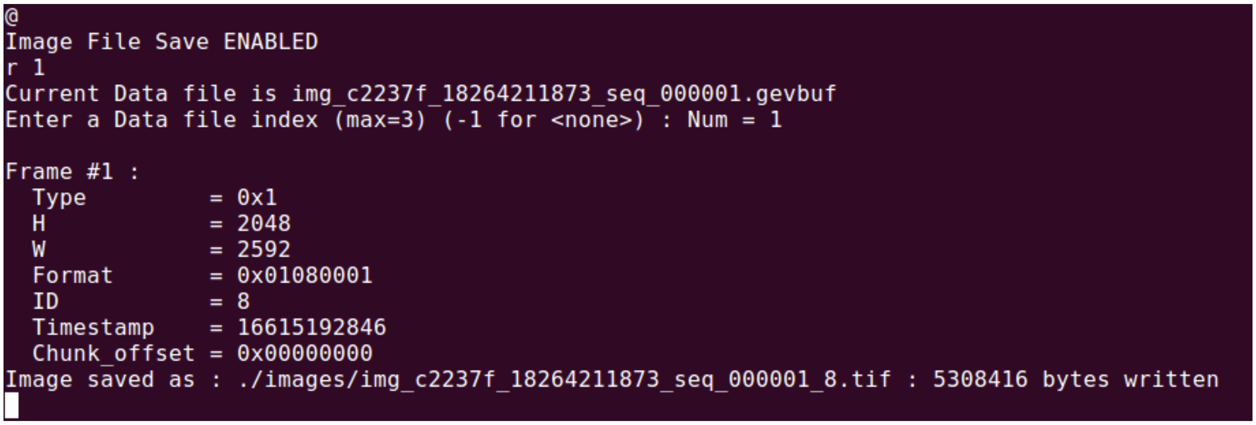
The restore_nano_data_demo reads ".gevbuf" files and restores individual images or sequences. Information for each frame in a sequence is displayed. The program unpacks packed images and decodes TurboDrive encoded frames. Bayer decoding can also be enabled (to convert to color).

The program lists the available files present in the directory where it is executed.



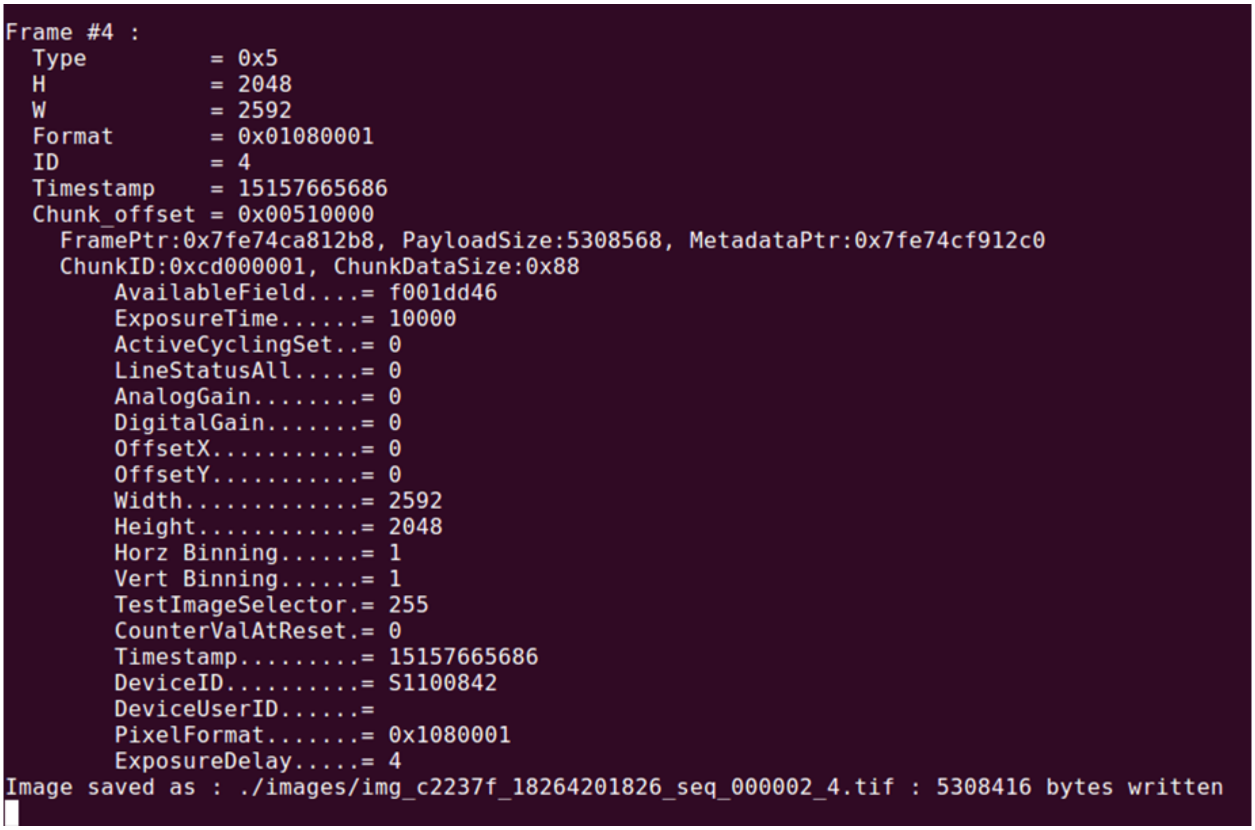
```
sapera@sapera-X8ST3:~/DALSA/GigeV/examples/archive_data$ ./restore_nano_data_demo

GigE Vision Library GenICam C Example Program (restore_data) (Sep 10 2018)
Copyright (c) 2018, DALSA. (No restrictions on further use)

  ============================

  Found 5 available raw data files "(*.gevbuf)" :
         0 :    img_c2237f_18264193541_000000004.gevbuf
         1 :    img_c2237f_18264192693_seq_000000.gevbuf
         2 :    img_c2237f_18264192693_seq_000001.gevbuf
         3 :    img_c2237f_18264193541_000000006.gevbuf
         4 :    img_c2237f_18264192693_seq_000002.gevbuf

  ============================

FILE SELECT : [R]=Read gevbuf file (by Index), [L]=List available gevbuf files
OPTIONS     : [@]=Enable/Disable Image Save (TIF) (Conversion to RGB if available)
MISC        : [Q]or[ESC]=end
```

Image saving can be enabled to store image / sequences as TIFF files (mono/RGB based on Bayer decoding setting when they are read). Each frame in a sequence is saved as a separate TIFF file. To read a file, use the "R" command, followed by the file index (to exit this command, use -1).



```
@
Image File Save ENABLED
r 1
Current Data file is img_c2237f_18264211873_seq_000001.gevbuf
Enter a Data file index (max=3) (-1 for <none>) : Num = 1

Frame #1 :
  Type         = 0x1
  H            = 2048
  W            = 2592
  Format       = 0x01080001
  ID           = 8
  Timestamp    = 16615192846
  Chunk_offset = 0x00000000
Image saved as : ./images/img_c2237f_18264211873_seq_000001_8.tif : 5308416 bytes written
```

The example reads metadata written using the Genie Nano layout; if the frame contains metadata, it is displayed in the console (if no metadata is present, 0x00000000 is returned).



```
Frame #4 :
  Type         = 0x5
  H            = 2048
  W            = 2592
  Format       = 0x01080001
  ID           = 4
  Timestamp    = 15157665686
  Chunk_offset = 0x00510000
    FramePtr:0x7fe74ca812b8, PayloadSize:5308568, MetadataPtr:0x7fe74cf912c0
    ChunkID:0xcd000001, ChunkDataSize:0x88
        AvailableField....= f001dd46
        ExposureTime......= 10000
        ActiveCyclingSet..= 0
        LineStatusAll.....= 0
        AnalogGain........= 0
        DigitalGain.......= 0
        OffsetX...........= 0
        OffsetY...........= 0
        Width.............= 2592
        Height............= 2048
        Horz Binning......= 1
        Vert Binning......= 1
        TestImageSelector.= 255
        CounterValAtReset.= 0
        Timestamp.........= 15157665686
        DeviceID..........= S1100842
        DeviceUserID......=
        PixelFormat.......= 0x1080001
        ExposureDelay.....= 4
Image saved as : ./images/img_c2237f_18264201826_seq_000002_4.tif : 5308416 bytes written
```

Action Command Example

The action_cmd_demo demonstrates how to send an action command to an available Genie Nano camera on a network. The demo uses action command 1 with the default device key, group key and group mask.

Camera features must be set to accept an action command as a trigger input. To do so, the nano_trigger_demo program can be used to configure a Genie Nano to trigger to acquire an image on receipt of action command 1. The camera must be in an active acquisition state (active snap or grab) to accept the action command trigger. Note that a separate terminal is required to run both programs (action_cmd_demo and nano_trigger_demo) concurrently

Alternately, the camera can be manually configured to accept an action command, using the GenICam trigger features supported by the device; refer to the device documentation for more information.

Firmware Update

The standard GenICam File Access features are used to update the camera firmware, if the camera supports firmware file write access. The GenICam Standard Features Naming Convention (SFNC) documentation is available at http://www.emva.org/standards-technology/genicam/.

The File Access Example demonstrates how to implement file access using the GigE-V Framework API feature access functions.

ⓘ	**Note:** After successfully writing (uploading) a new firmware file to the camera, the camera typically must be reset (using the GenICam DeviceReset command or power cycling the camera) to activate the new firmware; refer to the camera documentation for more information.

GigE-V Framework API

About GigE Vision

The GigE Vision standard describes a set of protocols that define access methods and capabilities for devices and applications alike. The main protocols applicable for GigE Vision cameras are GVCP (GigE Vision Control Protocol) and GVSP (GigE Vision Streaming Protocol).

The GigE-V Framework API supports the standard register and memory area access parts of GVCP as well as its asynchronous message channel. The API also supports image data acquisition from a device using GVSP.

The specific definitions of what is supported by a device are contained in the GigE Vision compliant XML file provided with the device. Starting with v2.0, the GigE-V Framework API library is able to read the XML file from the device, and associate a GenICam feature node tree with the device. For more information on how to use the XML based features see the code examples provided in this document and the example programs supplied with the API

The GigE-V Framework API builds on the GenICam GenApi, which is included in the GigE-V Framework API installation. For more information and documentation of the GenICam GenApi visit the EMVA (European Machine Vision Association) website: www.emva.org/standards-technology/genicam/.

What's New for Version 2.10

The GigE-V Framework API Version 2.10 adds support for the inclusion of metadata, or “chunk” data in the terminology of the GigE Vision and GenICam standards.

Data transfers from acquisition device include both the image data and corresponding metadata, which taken together form a “payload”. The “payload” concept allows for other types of data transfers from a device as these types become available through updates to this API for support of newer revisions of the GigE Vision standard.

To provide support "payloads", future standard updates, and to enhance ease of use, the following API changes have been adopted for Version 2.10:

- New names for functions were chosen to reinforce the concept of "frame" handling versus "image" handling. The functions can be used with the previous names also.

Previous Function Name (still supported)	**New Function Name**
GevFreeImageTransfer	GevFreeTransfer
GevStartImageTransfer	GevStartTransfer
GevStopImageTransfer	GevStopTransfer
GevAbortImageTransfer	GevAbortTransfer
GevWaitForNextImage	GevWaitForNextFrame
GetGetNextImage	GetGetNextFrame
GevReleaseImage	GevReleaseFrame
GevReleaseImageBuffer	GevReleaseFrameBuffer
GevQueryImagetransferStatus	GevQueryTransferStatus

- A new function, GevIntializeTransfer, to communicate the size of the allocated buffer to the API so the full payload can be safely stored, especially chunk data that is beyond the end of the image data.
- GenICam XML feature setup is now done automatically when the camera (or device) is opened. Manual XML handling can be restored by setting the "manual_xml_handling" entry in the GEVLIB_CONFIG_OPTIONS structure to TRUE (1) before opening a camera.
- Demos have been changed to use the mandatory "PayloadSize" feature to allocate buffers for payload frames. Usually the "PayloadSize" is the proper size for allocating buffers, however, in the case of data with packed pixels that get unpacked during transfer (Mono10Packed/Mono12Packed), the "PayloadSize" is smaller than the unpacked pixel image size. The example programs demonstrate how to handle this.
- New "passthru" mode to disable the automatic unpacking of packed pixel formats. The example programs provided in the archive_demo folder show how to use the “passthru” mode, demonstrating how to handle undisplayable packed formats
- The "id" field in the GEV_BUFFER_OBJECT structure, returned with the arrival of a frame, is now 64-bits for future support of a newer GigE Vision standard.

API Initialization and Configuration

This section describes the API functions to initialize the API and adjust the configuration parameters available to modify the API's behavior.

Member Function Overview

Function	Description
GevApiInitialize	Initializes the API.
GevApiUninitialize	Closes (un-initialize) the API.
GevApiGetLibraryConfigOptions, GevApiSetLibraryConfigOptions	Gets GigE-V Framework API library user configurable parameters. Sets GigE-V Framework API library user configurable parameters.

Member Function Descriptions

The following functions are members of the API Initialization and Configuration group.

GevApiGetLibraryConfigOptions, GevApiSetLibraryConfigOptions

```
GEV_STATUS GevGetLibraryConfigOptions(GEVLIB_CONFIG_OPTIONS *options);
GEV_STATUS GevSetLibraryConfigOptions(GEVLIB_CONFIG_OPTIONS *options);
```

Description

Obtains or updates the user configurable parameters that apply to the GigE-V Framework API library. The configurable options are contained in a data structure of type GEVLIB_CONFIG_OPTIONS and apply globally to the operation of the GigE-V Framework API library within the current application.

Parameters

options Pointer to a GEVLIB_CONFIG_OPTIONS structure:

```
typedef struct
{
    UINT32 version;
    UINT32 logLevel;
    UINT32 numRetries;
    UINT32 command_timeout_ms;
    UINT32 discovery_timeout_ms;
    UINT32 enumeration_port;
    UINT32 gvcp_port_range_start;
    UINT32 gvcp_port_range_end;
    UINT32 manual_xml_handling;
} GEVLIB_CONFIG_OPTIONS, *PGEVLIB_CONFIG_OPTIONS;
```

Structure Description

version The version of the API (it is read-only)

logLevel The current message severity logging level for informational messages. The logLevel can be set to select which messages are actually output. Possible values are:

GEV_LOG_LEVEL_OFF	No logging is performed
GEV_LOG_LEVEL_NORMAL	Fatal and error messages are enabled

GEV_LOG_LEVEL_ERRORS	Same as “NORMAL”
GEV_LOG_LEVEL_WARNINGS	Warning messages are also enabled
GEV_LOG_LEVEL_DEBUG	Debug messages are also enabled
GEV_LOG_LEVEL_TRACE	Trace messages are also enabled

The default value is GEV_LOG_LEVEL_NORMAL.

Messages are logged using GevPrint to print messages.
Messages can have the following levels of severity :

GEV_LOG_FATAL	For fatal errors.
GEV_LOG_ERROR	For general errors.
GEV_LOG_WARNING	For warnings
GEV_LOG_INFO	For informational purposes

Important: The more types of messages that are enabled, the more of a load is placed on the library to perform the logging. This can lead to degradation of performance in high data rate applications.

numRetries
Number of times a command is retried before giving up on the command and generating an error. This is to allow some tolerance for collisions and added traffic on the network interface connecting the PC to the camera. (The default value is 3)

command_timeout_ms
Milliseconds the library will wait for a response to a command before attempting to retry the command or, if the number of retries have been exhausted, failing the command.
(The default value is 2000 msecs)

discovery_timeout_ms
Milliseconds the library will wait for a response when querying the network for the presence of cameras. The number of retries setting also applies to the process of querying the presence of cameras. (The default value is 1000 msecs)

enumeration_port
IP (Internet Protocol) port on which the device enumeration/discovery will take place. This allows for the tuning of network port usage in a system. (The default value is 39999)

gvcp_port_range_start
gvcp_port_range_end
Start and end IP (Internet Protocol) port numbers for the range of ports used by the library for communicating with cameras. Port assignments are taken as needed, from this range and returned when they are no longer required. This allows for the tuning of network port usage in a system.
(The default range is 40000 to 49999)

manual_xml_handling
Flag to turn off automatic setup of XML features when a camera is opened. XML feature access must be performed manually if this is 1 (true).

Return Value

GEVLIB_OK

GevApiInitialize

```
GEV_STATUS GevApiInitialize(void);
```

Description

Initializes the API.

Return Value

GEVLIB_OK
GEVLIB_ERROR_INSUFFICIENT_MEMORY

GevApiUninitialize

```
GEV_STATUS GevApiUninitialize(void);
```

Description

Closes (un-initialize) the API.

Return Value

GEVLIB_OK

Automatic Camera Discovery

Functions are provided to perform automatic camera (device) discovery and enumeration.

Member Function Overview

Function	Description
GevDeviceCount	Function used to query the number of cameras detected in the system.
GevGetCameraList	Function returns a list of cameras detected as present in the system.

Member Function Descriptions

The following functions are members of the Automatic Camera Discovery group.

GevDeviceCount

```
int GevDeviceCount(void);
```

Description

Queries the number of cameras detected in the system.
Note: A number of factors determine whether connected cameras are seen in the system. Most notably, the camera and network interface card (NIC) must be on the same IPV4 subnet.

Return Value

The return value is the number of cameras visible in the system.

GevGetCameraList

```
GEV_STATUS GevGetCameraList(GEV_CAMERA_INFO *cameras, int maxCameras,
                            int *numCameras);
```

Description

Returns a list of cameras detected as present in the system.

Parameters

cameras	Pointer to an array of GEV_CAMERA_INFO structures, allocated by the caller, to contain information for the cameras detected in the system.
maxCameras	Maximum number of entries in the array of GEV_CAMERA_INFO structures passed in the 'cameras' parameter.
numCameras	Pointer to contain the number of cameras actually detected in the system. (Note: The number of cameras found can be larger than the number of entries in the 'cameras' array. In this case, only 'maxCameras' entries are returned in the array. The total number of cameras in the system is returned in 'numCameras'.)

Return Value

GEVLIB_OK.

Connecting to a Camera

After cameras are detected by the system, they can be connected to and accessed via a ‘handle’ (of type GEV_CAMERA_HANDLE). GigE Vision makes a distinction between classes of connection. Primary control connections and secondary control connections are supported.

A connection using the primary control channel to a camera is able to control all aspects of the camera function including its streaming interface and its asynchronous message channel. If this connection is exclusive, no other connections can be made to the camera. If the primary control channel is not being used in an exclusive mode, a secondary control channel can be opened and the camera queried for monitor access. Applications using the secondary control channel can only read from the camera and are used only for monitoring.

The following functions provide a means to create the camera handle for device access. These functions are compatible for use in both C and C++ language application programs.

ℹ **Note:** In all cases, the camera device and the NIC card must share the same IP subnet mask.

Member Function Overview

Function	Description
GevOpenCamera	Creates a camera handle for accessing a camera.
GevOpenCameraByAddress	Creates a camera handle for accessing a camera identified by a its IP address.
GevOpenCameraByName	Creates a camera handle for accessing a camera identified by a its user name.
GevOpenCameraBySN	Creates a camera handle for accessing a camera identified by a its serial number.
GevGetCameraInterfaceOptions, GevSetCameraInterfaceOptions	Obtains the user configurable parameters. Updates the user configurable parameters.
GevGetCameraInfo	Obtains a pointer to the GEV_CAMERA_INFO structure.
GevCloseCamera	Closes a previously opened camera handle and terminates access.

Member Function Descriptions

The following functions are members of the Camera Access group.

GevCloseCamera

```
GEV_STATUS GevCloseCamera(GEV_CAMERA_HANDLE *handle);
```

Description

Closes a previously opened camera handle and terminates access to the camera from the application.

Parameters

handle	Pointer to a GEV_CAMERA_HANDLE type to receive the allocated handle, used to access the camera.

Return Value

GEV_STATUS	Possible values are: GEVLIB_ERROR_INVALID_HANDLE GEVLIB_OK

GevGetCameraInfo

```
GEV_CAMERA_INFO *GevGetCameraInfo(GEV_CAMERA_HANDLE handle);
```

Description

Obtains a pointer to the GEV_CAMERA_INFO structure stored internally in the camera handle.

Parameters

handle	Pointer to a GEV_CAMERA_HANDLE type to receive the allocated handle, used to access the camera.

Return Value

Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE

GevGetCameraInterfaceOptions, GevSetCameraInterfaceOptions

```
GEV_STATUS GevGetCameraInterfaceOptions(GEV_CAMERA_HANDLE handle,
                                        GEV_CAMERA_OPTIONS *options);

GEV_STATUS GevSetCameraInterfaceOptions(GEV_CAMERA_HANDLE handle,
                                        GEV_CAMERA_OPTIONS *options);
```

Description

These functions are used to obtain and update the user configurable parameters that apply to the camera connection through the camera handle. The configurable options are contained in a data structure of type GEV_CAMERA_OPTIONS and apply only to the camera accessed through the specific camera handle.

Parameters

handle Pointer to a GEV_CAMERA_HANDLE type to receive the allocated handle, used to access the camera.

options Pointer to a data structure of type GEV_CAMERA_OPTIONS, allocated by the caller, that contains the parameters associated with the underlying camera handle. This type is defined as

```
typedef struct
{
      UINT32 numRetries;
      UINT32 command_timeout_ms;
      UINT32 heartbeat_timeout_ms;
      UINT32 streamPktSize;
      UINT32 streamPktDelay
      UINT32 streamNumFramesBuffered;
      UINT32 streamMemoryLimitMax;
      UINT32 streamMaxPacketResends;
      UINT32 streamFrame_timeout_ms;
      INT32  streamThreadAffinity;
      INT32  serverThreadAffinity;
      UINT32 msgChannel_timeout_ms;
      UINT32 enable_passthru_mode;
} GEV_CAMERA_OPTIONS, *PGEV_CAMERA_OPTIONS;
```

Structure Description

numRetries — Number of times a command is retried before giving up on the command and generating an error. This is to allow some tolerance for collisions and added traffic on the network interface connecting the PC to the camera. (The default value is 3)

command_timeout_ms — Milliseconds the library waits for a response to a command before attempting to retry the command or, if the number of retries have been exhausted, failing the command. (The default value is 2000 msecs)

heartbeat_timeout_ms — Milliseconds the library and camera waits for contact between the application and the camera before the camera decides that the application is unresponsive and releases the connection. (The default value is 10000 msecs)

streamPktSize — Size, in bytes, of the data packets used for streaming data from the camera. This value is determined algorithmically when the camera is opened and can be overridden by setting a new value using this parameter. The new value must be less than the NIC MTU (maximum transmission unit) size.

streamPktDelay Time delay, in microsecond, between packets sent from the camera. It can be used to adjust the performance of the packet streaming on busy network segments. (The default is 0).

streamNumFramesBuffered Sets the number of frames that can be buffered concurrently in an internal list. These frames remain in the list until their acquisition is completed either successfully, or with some error condition caused by problems encountered during the acquisition. With a good connection to the camera, the number of frames actually being buffered at any given time is one. The default is 4. The minimum is 2.

streamMemoryLimitMax Maximum amount of memory to use (puts an upper limit on the number of frames to buffer).

streamMaxPacketResends Maximum number of packet resends to allow for a frame (defaults to 100).

The time, in milliseonds, that a frame is active in the internal buffering list before it is completed with a timeout error. The time is measured from the reception of the first packet for the frame from the camera. The default is 1000 ms.

streamFrame_timeout_ms Milliseconds, following the reception of the start of a frame, that the API waits for a frame to be completed. If this time is exceeded, the frame is delivered to the application with the status member of the GEVBUF_HEADER structure set to GEV_FRAME_STATUS_TIMEOUT.

streamThreadAffinity CPU index (0 to 1023) used to specify a particular CPU on which to create the streaming packet receive thread when running a multi-CPU system. A value of “-1” allows the thread to be created on whatever default CPU the OS chooses. A value that is larger than the number of CPUs in a system is treated as if it is “-1”. (The default is -1)

serverThreadAffinity CPU index (0 to 1023) used to specify a particular CPU on which to create the high performance packet server thread when running a multi-CPU system. The packet server thread reads packets from the PF_PACKET socket interface which intercepts network data before it is written into the systems network stack. A value of “-1” allows the thread to be created on whatever default CPU the OS chooses based on its (fairly reasonable) load balancing algorithm. A value that is larger than the number of CPUs in a system is treated as if it is “-1”. (The default is -1)

msgChannel_timeout_ms Milliseconds that the asynchronous messaging thread waits during its periodic checks for asynchronous messages from the camera. (The default is 1 second)

enable_passthru_mode Zero (default) to enable automatic unpacking of packed pixel formats and decoding of TurboDrive formats.

Non-zero for passthru mode.

Return Value

GEV_STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEV_STATUS_NULL_PTR

GevOpenCamera

```
GEV_STATUS GevOpenCamera(GEV_CAMERA_INFO *device, GevAccessMode mode,
                         GEV_CAMERA_HANDLE *handle);
```

Description

Creates a camera handle for accessing a camera identified by an input camera information structure (type GEV_CAMERA_INFO).

Parameters

device	Pointer to a GEV_CAMERA_INFO structure, allocated by the caller, passed in to identify the camera device to open.
mode	Required access mode. The available values are: GevExclusiveMode : Exclusive R/W access to the camera. GevMonitorMode : Shared Read-only access to the camera. GevControlMode : Shared R/W access to the camera. The most commonly used mode, for user imaging applications, is GevExclusiveMode.
handle	Pointer to a GEV_CAMERA_HANDLE type Receives the allocated handle to be used to access the camera.

Return Value

GEV_STATUS	Possible values are: GEVLIB_ERROR_API_NOT_INITIALIZED GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_INSUFFICIENT_MEMORY GEVLIB_ERROR_NO_CAMERA GEV_STATUS_ACCESS_DENIED

GevOpenCameraByAddress

```
GEV_STATUS GevOpenCameraByAddress(unsigned long ip_address,GevAccessMode mode,
                                  GEV_CAMERA_HANDLE *handle);
```

Description

Creates a camera handle for accessing a camera identified by a camera’s IP address.

Parameters

ip_address	32-bit IP address for a camera, as a number. For example, 192.168.1.10 is 0xC0A8010A.
mode	Required access mode. The available values are: GevExclusiveMode : Exclusive R/W access to the camera. GevMonitorMode : Shared Read-only access to the camera. GevControlMode : Shared R/W access to the camera. The most commonly used mode for user imaging applications is GevExclusiveMode.
handle	Pointer to a GEV_CAMERA_HANDLE type to receive the allocated handle to be used to access the camera.

Return Value

GEV_STATUS	Possible values are: GEVLIB_ERROR_API_NOT_INITIALIZED GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_INSUFFICIENT_MEMORY GEVLIB_ERROR_NO_CAMERA GEV_STATUS_ACCESS_DENIED

GevOpenCameraByName

```
GEV_STATUS GevOpenCameraByName (char *name, GevAccessMode mode,
                                 GEV_CAMERA_HANDLE *handle);
```

Description

Creates a camera handle for accessing a camera identified by a camera’s user defined name. The user defined name is a string that can be programmed into the camera for use in identifying multiple cameras.

Parameters

name A character string (16 characters max) that will be used to match the user defined name string contained in a camera connected on the system.

mode The required access mode. The available values are:

GevExclusiveMode : Exclusive R/W access to the camera.
GevMonitorMode : Shared Read-only access to the camera.
GevControlMode : Shared R/W access to the camera.

The most commonly used mode for user imaging applications is GevExclusiveMode.

handle Pointer to a GEV_CAMERA_HANDLE type
to receive the allocated handle to be used to access the camera.

Return Value

GEV_STATUS Possible values are:

GEVLIB_ERROR_API_NOT_INITIALIZED
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_INSUFFICIENT_MEMORY
GEVLIB_ERROR_NO_CAMERA
GEV_STATUS_ACCESS_DENIED

GevOpenCameraBySN

```
GEV_STATUS GevOpenCameraBySN(char *sn, GevAccessMode mode,
                             GEV_CAMERA_HANDLE *handle);
```

Description

Creates a camera handle for accessing a camera identified by a camera's serial number. The serial number is represented as a string that is programmed into the camera, by the manufacturer, to identify a particular camera unit.

Parameters

sn A character string (16 characters max) that matches the serial number string contained in a camera connected on the system.

mode The required access mode. The available values are:

GevExclusiveMode	: Exclusive R/W access to the camera.
GevMonitorMode	: Shared Read-only access to the camera.
GevControlMode	: Shared R/W access to the camera.

The most commonly used mode, for user imaging applications, is GevExclusiveMode.

handle Pointer to a GEV_CAMERA_HANDLE type to receive the allocated handle used to access the camera.

Return Value

GEV_STATUS Possible values are:

GEVLIB_ERROR_API_NOT_INITIALIZED
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_INSUFFICIENT_MEMORY
GEVLIB_ERROR_NO_CAMERA
GEV_STATUS_ACCESS_DENIED

Camera GenICam Feature Access

This section describes the functions provided for accessing camera features defined by the GenICam compatible definitions obtained from the vendor supplied XML data corresponding to the camera. These functions are compatible for use in both C and C++ language application programs.

Member Function Overview

Function	Description
GevGetFeatureValue	Retrieves the value of a GenICam feature, as well as its type, by name.
GevSetFeatureValue	Sets the value of a GenICam feature, by name.
GevGetFeatureValueAsString	Retrieves a string representation of the value of a GenICam feature, as well as its type, by name.
GevSetFeatureValueAsString	Sets the value of a GenICam feature, by name, via a string representation of the value.
GevGetFeatureNodeMap	Retrieves a pointer to a feature node map from a handle.

Member Function Descriptions

The following functions are members of the Camera GenICam Feature Access group.

GevGetFeatureNodeMap

```
void * GevGetFeatureNodeMap(GEV_CAMERA_HANDLE handle);
```

Description

Returns, as a void pointer, a pointer to a GenApi::CNodeMapRef object that was previously associated with the camera handle by a call to GevConnectFeatures. This allows the pointer to be retrieved from the API for use in cases where only the camera handle is available.
The feature node map pointer is required if the application program wants to directly access the underlying GenApi:: interface provided by the GenICam standards group in order to access additional functionality not provided by the GigE-V Framework for Linux.

Parameters

handle	Handle to the camera.

Return Value

A non-NULL pointer on success. A NULL pointer on error or incomplete initialization.

Note: The feature node map is automatically set up when a device is opened. If the pointer returned is NULL, then there was either an error opening the device or the GEVLIB_CONFIG_OPTIONS for the library was modified to enable manual XML handling and the XML to device connection needs to be established manually.

GEVLIB_OK on success.

GevGetFeatureValue

```
GEV_STATUS GevFeatureValue (GEV_CAMERA_HANDLE handle, const char *feature_name,
                            int *feature_type, int value_size, void *value);
```

Description

Retrieves the value of a feature as well as its type.
This function is intended to be used from C code, where the GenApi object class accesses are not supported.

Note : The corresponding GenApi::CNodeMapRef object must already be associated with the camera handle via call to GevConnectFeatures.

Parameters

handle Handle to the camera.

feature_name String containing the name of the feature to be accessed.

feature_type Pointer to storage to the feature type being returned. This is the integer version of the GenApi::EInterfacetype associated with the feature node accessed by name. The valid values are :

```
GENAPI_UNUSED_TYPE    = 1   for intfIBase/intfIValue/intfICategory that
                            are not accessible from C code.
GENAPI_INTEGER_TYPE   = 2   for GenApi::EInterfaceType intfIInteger
GENAPI_BOOLEAN_TYPE   = 3   for GenApi::EInterfaceType intfIBoolean
GENAPI_COMMAND_TYPE   = 4   for GenApi::EInterfaceType intfICommand
GENAPI_FLOAT_TYPE     = 5   for GenApi::EInterfaceType intfIFloat
GENAPI_STRING_TYPE    = 6   for GenApi::EInterfaceType intfIString
GENAPI_REGISTER_TYPE  = 7   for GenApi::EInterfaceType intfRegister
GENAPI_ENUM_TYPE      = 9   for GenApi::EInterfaceType intfIEnum
GENAPI_ENUMENTRY_TYPE = 10  for GenApi::EInterfaceType intfIEnumEntry
```

value_size Size, in bytes, of the storage pointed to by “value” that receives the data contained at the feature node being accessed.

value Pointer to storage at which to return the data read from the feature node.

Return Value

GEVLIB_OK on success.

GevGetFeatureValueAsString

```
GEV_STATUS GevGetFeatureValueAsString (GEV_CAMERA_HANDLE handle, const char
                                       *feature_name, int *feature_type, int
                                       value_string_size, char *value_string);
```

Description

Reads the value of a feature and returns it as a string representation.
This function is useful in C and C++ code, especially for representing feature names and values in a GUI program.

Note : The corresponding GenApi::CNodeMapRef object must already be associated with the camera handle via call to GevConnectFeatures.

Parameters

handle Handle to the camera.
feature_name String containing the name of the feature to be accessed.
feature_type Pointer to storage to the feature type being returned. This is the integer version of the GenApi::EInterfacetype associated with the feature node accessed by name. The valid values are :

```
GENAPI_UNUSED_TYPE    = 1   for intfIBase/intfIValue/intfICategory that
                            are not accessible from C code.
GENAPI_INTEGER_TYPE   = 2   for GenApi::EInterfaceType intfIInteger
GENAPI_BOOLEAN_TYPE   = 3   for GenApi::EInterfaceType intfIBoolean
GENAPI_COMMAND_TYPE   = 4   for GenApi::EInterfaceType intfICommand
GENAPI_FLOAT_TYPE     = 5   for GenApi::EInterfaceType intfIFloat
GENAPI_STRING_TYPE    = 6   for GenApi::EInterfaceType intfIString
GENAPI_REGISTER_TYPE  = 7   for GenApi::EInterfaceType intfRegister
GENAPI_ENUM_TYPE      = 9   for GenApi::EInterfaceType intfIEnum
GENAPI_ENUMENTRY_TYPE = 10  for GenApi::EInterfaceType intfIEnumEntry
```

value_string_size Size, in bytes, of the storage pointed to by “value_string” that is to contain string version of the feature value.
value_string Pointer to storage at which string version of the value is copied on return.

Return Value

GEVLIB_OK on success.

GevSetFeatureValue

```
GEV_STATUS GevSetFeatureValue (GEV_CAMERA_HANDLE handle, const char *feature_name,
                               int value_size, void *value)
```

Description

Writes the value of a feature. This function is intended to be used from C code, where the GenApi object class accesses are not supported.

Note : The corresponding GenApi::CNodeMapRef object must already be associated with the camera handle via call to GevConnectFeatures.

Parameters

handle Handle to the camera.
feature_name String containing the name of the feature to be accessed.
value_size Size, in bytes, of the storage pointed to by “value” that contains the data to be written to the feature node being accessed.
Note: The feature node already knows the type of data that it expects.
value Pointer to storage at which the data to be written is located.

Return Value

GEVLIB_OK on success.

GevSetFeatureValueAsString

```
GEV_STATUS GevSetFeatureValueAsString (GEV_CAMERA_HANDLE handle, const char
                                      *feature_name, char *value_string);
```

Description

Writes the value of a feature using its string representation.
This function is useful in C and C++ code, especially for representing feature names and values in a GUI program.

Note : The corresponding GenApi::CNodeMapRef object must already be associated with the camera handle via call to GevConnectFeatures.

Parameters

handle	Handle to the camera.
feature_name	String containing the name of the feature to be accessed.
value_string_size	Size, in bytes, of the storage pointed to by "value_string" that contains the string version of the feature value.
value_string	Pointer to storage for the string version of the value being written.

Return Value

Example C Code : Open camera and access features

```
GEV_DEVICE_INTERFACE  pCamera[MAX_CAMERAS] = {0};
int numCamera = 0;
int camIndex = 0;
int type;
GEV_CAMERA_HANDLE handle = NULL;
char xmlFileName[MAX_PATH] = {0};
UINT32 height, width;
UINT64 size;
char pixelfmt[64] = {0};

// Get camera list.
GevGetCameraList( pCamera, MAX_CAMERAS, &numCamera);

// Open the camera you want
GevOpenCamera( &pCamera[camIndex], GevExclusiveMode, &handle);

// Get the image dimensions, payload size, and format.
GevGetFeatureValue( handle, “Height”, &type, sizeof(height), &height);
GevGetFeatureValue( handle, “Width”, &type, sizeof(width), &width);
GevGetFeatureValue( handle, “PayloadSize”, &type, sizeof(size), &size);

GevGetFeatureValueAsString( handle, “PixelFormat”, &type, sizeof(pixelfmt), pixelfmt);
```

Example C++ Code: Open camera and set up access to the GenICam Feature Node Map for GenApi access.

```
GEV_DEVICE_INTERFACE  pCamera[MAX_CAMERAS] = {0};
int numCamera = 0;
int camIndex = 0;
GEV_CAMERA_HANDLE handle = NULL;
UINT64 payload_size;
UINT32 pixel_format;

// Get camera list.
GevGetCameraList( pCamera, MAX_CAMERAS, &numCamera);

// Open the camera you want
GevOpenCamera( &pCamera[camIndex], GevExclusiveMode, &handle);

// Get the payload parameters (for buffer memory allocation)
GevGetPayloadParameters( handle,  &payload_size, &pixel_format);

// Get GenICam FeatureNodeMap object and access the camera features.
 GenApi::CNodeMapRef *Camera = \
          static_cast<GenApi::CNodeMapRef*>(GevGetFeatureNodeMap(handle));

 < … GenApi access to features from here on via Camera object … >
```

Camera GenICam Feature Access – Manual Setup

This section describes the functions provided for manually setting up access to XML-defined GenICam features. The functions show how to retrieve the XML definitions from a camera, how to instantiate a GenICam feature node tree, how to associate/connect the node tree to a camera.

These functions are intended to be used in embedded systems where their might not be disk storage present to store the XML definitions to a file.

(The “manual_xml_handling” member of the GEVLIB_CONFIG_OPTIONS structure should be non-zero (true) in order to circumvent the automatic setup of XML feature access and allow it to be performed manually).

Member Function Overview

Function	Description
GevGetGenICamXML_FileName	Retrieves the name of the file (if any) used to initialize the GenICam features.
GevInitGenICamXMLFeatures	Initializes access to GenICam features based on the XML file in the camera.
GevInitGenICamXMLFeatures_FromFile	Initializes access to GenICam features based on an XML file on disk.
GevInitGenICamXMLFeatures_FromData	Initializes access to GenICam features based on XML data in a string.
Gev_RetrieveXMLData	Retrieves the XML data from the camera.
Gev_RetrieveXMLFile	Retrieves the XML file from the camera.
GevConnectFeatures	Connects a feature node map to a camera handle.

Member Function Descriptions

The following functions are members of the Camera GenICam Feature Access (Manual Setup) group.

Gev_RetrieveXMLData

```
GEV_STATUS Gev_RetrieveXMLData (GEV_CAMERA_HANDLE handle, int size, char
                                *xml_data, int *num_read, int *data_is_compressed);
```

Description

Retrieves XML data used for the camera from the camera itself. The data is returned in the location pointed to by the input data buffer. The number of bytes read from the camera is also returned. Note: If the input buffer pointer is NULL, the function returns the required size of the XML data buffer.

Parameters

handle	Handle to the camera.
size	Size (in bytes) of the XML data buffer passed in.
xml_data	Pointer to storage to hold XML data read from the camera.
num_read	Pointer to hold the number of bytes read from the camera. If the “xml_data” pointer is NULL, the required buffer size is returned here.
data_is_compressed	Pointer to hold a flag indicating if the returned XML data is compressed (1 for true) or not (0 for false)

Return Value

GEVLIB_OK on success.

Gev_RetrieveXMLFile

```
GEV_STATUS Gev_RetrieveXMLFile (GEV_CAMERA_HANDLE handle, char *filename, int size,
                                BOOL force_download);
```

Description

Retrieves the name of the XML file to use for the camera. If the XML file has not yet been downloaded from the camera, it is downloaded and stored in the subdirectory 'xml/<manufacturer>' of the installation directory pointed to by the GIGEV_XML_DOWNLOAD environment variable.

If the GIGEV_XML_DOWNLOAD environment variable is not set, the XML file isstored in the 'xml/<manufacturer>' subdirectory of the program executing.

Generally, once the XML file is already on the local disk, it is not downloaded again. If the "force_download" flag is set, the XML file is downloaded, regardless of whether it is on the disk or not.

Parameters

handle	Handle to the camera.
filename	Pointer to a string to receive the XML file name (as stored in the camera)
size	Number of bytes available to store the file name in the filename string.
force_download	If TRUE, the XML file is always downloaded from the camera overwriting the file on disk. If FALSE, the XML file is downloaded from the camera only if it does not exist on disk.

Return Value

GEVLIB_OK on success.

GevConnectFeatures

```
GEV_STATUS GevConnectFeatures(GEV_CAMERA_HANDLE handle, void *featureNodeMap);
```

Description

Connects a GenApi::CNodeMapRef object with the device port associated with the camera handle. The CNodeMapRef object is passed in as a void pointer.
Note: There is no way for the API to verify, ahead of time, that the void pointer provided is indeed a pointer to a valid GenApi::CNodeMapRef object. An error is returned, however, if the GenApi environment throws an exception while attempting to use the pointer as a GenApi::CNodeMapRef for the connection to the device port

Parameters

handle	Handle to the camera.
featureNodeMap	Void pointer that is assumed to point to a GenApi::CNodeMapRef object that is to be associated with the input camera handle. The feature node map is accessed to initialize internal access to mandatory features as well as some useful ones.

Return Value

GEVLIB_OK on success.

GevGetGenICamXML_FileName

```
GEV_STATUS GevGetGenICamXML_FileName(GEV_CAMERA_HANDLE handle, int size,
                                     char *xmlFileName);
```

Description

Returns the full path name of the XML file that was used to create the GenApi::CNodeMapRef object containing the feature tree for the camera.

Note: If the XML data is from a string/data buffer, or from the camera but not stored on disk, then the returned file name is blank.

Parameters

handle	Handle to the camera.
size	Size (in bytes) allocated to hold the full path name of the XML file currently in use.
xmlFileName	The full path name of the XML file that is in use.

GevInitGenICamXMLFeatures

```
GEV_STATUS GevInitGenICamXMLFeatures(GEV_CAMERA_HANDLE handle, BOOL updateXMLFile);
```

Description

Retrieves the GenICam XML file from the camera and uses it to initialize internal access to the GenICam GenApi via an internal GenApi::CNodeMapRef object connected to the camera. Optionally, the XML file read from the camera is stored to disk.

Parameters

handle	Handle to the camera.
updateXMLFile	The GenApi:CNodeMapRef object is created from the XML data retrieved from the camera accessed via the camera handle. If this flag is false, the XML file is not stored to disk. If this flag is true, the XML file is stored to disk. The location (path) to the stored XML files will be relative to the GIGEV_XML_DOWNLOAD environment variable. The path will be: $GIGEV_XML_DOWNLOAD/xml/download. If that location is not writable by the application, the XML file will be stored in the “current” directory that the executable is running in.

Return Value

GEVLIB_OK on success

GevInitGenICamXMLFeatures_FromData

```
GEV_STATUS GevInitGenICamXMLFeatures_FromData (GEV_CAMERA_HANDLE handle, int size,
                                                void *xmlDataBuffer);
```

Description

Initializes internal access to the GenICam GenApi, using the GenICam XML data string contained in the xmlDataBuffer, via an internal GenApi::CNodeMapRef object connected to the camera.

Parameters

handle	Handle to the camera.
size	Size (in bytes) of the XML data string passed in (including the terminating NULL '\0'). To aid in detection of an invalid XML definition.
xmlDataBuffer	Data array (string) containing a properly qualified XML definition for creating the GenApi::CNodeMapRef object.

GevInitGenICamXMLFeatures_FromFile

```
GEV_STATUS GevInitGenICamXMLFeatures_FromFile(GEV_CAMERA_HANDLE handle,
                                               char *xmlFileName);
```

Description

Initializes internal access to the GenICam GenApi, using the GenICam XML file identified by name, via an internal GenApi::CNodeMapRef object connected to the camera.

Parameters

Handle	Handle to the camera.
xmlFileName	Full path name of the XML file used to create the GenAPI::CNodeMapRef object.

GenICam GenApi Feature Access through XML

This section describes how to use the GenApi feature node tree directly. Code examples, in C++, are given to aid in using the GenApi interface provided by the GenICam standard libraries.

Example C++ Code: Simplified Access to GenICam Feature Node Map

```
 GEV_DEVICE_INTERFACE  pCamera[MAX_CAMERAS] = {0};
 int numCamera = 0;
 int camIndex = 0;
 GEV_CAMERA_HANDLE handle = NULL;

 // Get camera list.
 GevGetCameraList( pCamera, MAX_CAMERAS, &numCamera);

 // Open the camera you want
 GevOpenCamera( &pCamera[camIndex], GevExclusiveMode, &handle);


 // Set up feature access using the XML file retrieved from the camera
GenApi::CNodeMapRef *Camera = \
          static_cast<GenApi::CNodeMapRef*>(GevGetFeatureNodeMap(handle));

  < … GenApi access to features from here on via pointer to Camera object … >
```

Example C++ Code: Retrieve a Pointer to the GenICam Feature Node Map and Use GenApi Directly

```
GenApi::CNodeMapRef *pCamera = \
          static_cast<GenApi::CNodeMapRef*>(GevGetFeatureNodeMap(handle));

if (pCamera)
{
    // Access the features (by pointer)
    GenApi::CIntegerPtr ptrIntNode = pCamera->_GetNode("Width");
    UINT32 width = (UINT32) ptrIntNode->GetValue();
    ptrIntNode = pCamera->_GetNode("Height");
    UINT32 height = (UINT32) ptrIntNode->GetValue();

    GenApi::CEnumerationPtr ptrEnumNode = pCamera->_GetNode("PixelFormat") ;
    format = (UINT32)ptrEnumNode->GetIntValue();
}
```

For developers wanting to handle the management of the XML and feature node map themselves, either to wrap it all in an application level class or to alter the default handling of the XML, the following code examples are provided.

Example C++ Code: Read XML as Data and Manually Instantiate a GenICam Feature Node Map for the Camera

```
GEV_DEVICE_INTERFACE  pCamera[MAX_CAMERAS] = {0};
GEVLIB_CONFIG_OPTIONS options = {0};
int numCamera = 0;
int camIndex = 0;
GEV_CAMERA_HANDLE handle = NULL;
GenApi::CNodeMapRef Camera;

 // Set manual XML handling mode for the library.
 GevGetLibraryConfigOptions( &options);
 options. manual_xml_handling = 1;
 GevSetLibraryConfigOptions( &options);

// Get camera list.
GevGetCameraList( pCamera, MAX_CAMERAS, &numCamera);

// Open the camera you want
GevOpenCamera( &pCamera[camIndex], GevExclusiveMode, &handle);

// Retrieve the XML data from the camera
{
    int xmlFileSize = 0;
    char *pXmlData;
    BOOL compressed_data = 0;

   Gev_RetrieveXMLData( handle, 0, NULL, &xmlFileSize);
   xmlFileSize = (xmlFileSize + 3) & (~3));
   pXmlData = (char *)malloc( xmlFileSize + 1);
   Gev_RetrieveXMLData( handle, xmlFileSize, pXmlData, &xmlFileSize, &compressed_data);
    pXmlData[xmlFileSize ] = 0;
    GenICam::gcstring xmlStr( pXmlData );

  // Generate the feature node map from the XML data.
  if (compressed_data)
  {
     Camera._LoadXMLFromZIPData(xmlStr);
  }
  else
  {
     Camera._LoadXMLFromString(xmlStr);
  }
  free(pXmlData);
}

// Connect the camera to the feature map
GevConnectFeatures( handle, (void *)&Camera);

< … GenApi access to features from here on via Camera object … >
```

Example C++ Code: Use a previously stored XML File and Manually Instantiate a GenICam Feature Node Map for the Camera

```
GEV_DEVICE_INTERFACE  pCamera[MAX_CAMERAS] = {0};
GEVLIB_CONFIG_OPTIONS options = {0};
int numCamera = 0;
int camIndex = 0;
GEV_CAMERA_HANDLE handle = NULL;
GenApi::CNodeMapRef Camera;

 // Set manual XML handling mode for the library.
 GevGetLibraryConfigOptions( &options);
 options. manual_xml_handling = 1;
 GevSetLibraryConfigOptions( &options);

// Get camera list.
GevGetCameraList( pCamera, MAX_CAMERAS, &numCamera);

// Open the camera you want
GevOpenCamera( &pCamera[camIndex], GevExclusiveMode, &handle);

// Set up the XML data from a previously saved file.{
    char xmlFileName[MAX_PATH] = ”TeledyneDALSA_Nano-IMX249_Mono_2M.xml”;
      Camera._LoadXMLFromFile( xmlFileName );
}

// Connect the camera to the feature map
GevConnectFeatures( handle, (void *)&Camera);

 < … GenApi access to features from here on via Camera object … >
```

Image Frame Acquisition

This section describes functions that perform acquisition of image frames. Frames include both image data and metadata.

Member Function Overview

Function	Description
GevGetPayloadParameters	Retrieves the mandatory, payload specific information that identifies the size and format of data that the device will stream.
GevInitializeTransfer	Initializes a streaming transfer to the list of buffers indicated.
GevWaitForNextFrame	Waits for the next frame object to be acquired and returns its pointer.
GevGetNextFrame	Waits for the next frame object to be acquired and returns its pointer.
GevReleaseFrameBuffer	Releases a frame object back to the acquisition process for re-use.
GevReleaseFrame	Releases a frame object back to the acquisition process for re-use.
GevStopTransfer	Stops the streaming transfer.
GevAbortTransfer	Stops the streaming transfer immediately.
GevFreeTransfer	Frees a streaming transfer to the list of buffers indicated.

Structure Definition: GEVBUF_HEADER

The image buffer header structure is defined as follows:

```
typedef struct
{

    UINT32 payload_type;   // Type of payload received (Image(1), Raw(2), (etc..)
    UINT32 state;          // Full/empty state for payload buffer (tag used for buffer cycling)
    INT32  status;         // Frame Status as GEV_FRAME_STATUS_* (see below)
    UINT32 timestamp_hi;   // Most 32 significant bit of the timestamp (for legacy code)
    UINT32 timestamp_lo;   // Least 32 significant bit of the timestamp (for legacy code)
    UINT64 timestamp;      // 64bit version of timestamp for payload
    UINT64 recv_size;      // Received size of entire payload (including all appended "chunk"
                           //(metadata) information) .
    UINT64 id;             // Block id for the payload (starts at 1, may wrap to 1 at 65535).
                           // Image specific payload entries.
    UINT32 h;              // Received height (lines) for an image payload.
    UINT32 w;              // Received width (pixels) for an image payload.
    UINT32 x_offset;       // Received x offset for origin of ROI for an image payload_type.
    UINT32 y_offset;       // Received y offset for origin of ROI for an image payload_type.
    UINT32 x_padding;      // Received x padding bytes for an image payload_type
    UINT32 y_padding;      // Received y padding bytes for an image payload_type
    UINT32 d;              // Received pixel depth (bytes per pixel
    UINT32 format;         // Received GigE Vision pixel format for image types.
    PUINT8 address;        // Address of the "payload_type" data
                           //
                           // New entries for non-image payload types
                           //
    PUINT8 chunk_data;     // Address of "chunk" data (metadata) associated with the received
                           // payload (NULL if no "chunk" data (metadata) is available).
                           // The "chunk_data" address is provided here as a shortcut. It
                           // address immediately following the end of "paylod_type" data)
    UINT32 chunk_size;     // The size of the chunk_data (uncompressed). Zero if no "chunk" data
                           // (metadata) is available.
                           // The "chunk_size" is provided as a helper for decoding raw
                           // TurboDrive compressed data in passthru mode)
                           //
   char  filename[256];    // Name of file for payload type "file" (0 terminated string, 255
                           //characters maximum system limit in Linux).
} GEVBUF_HEADER, *PGEVBUF_HEADER;
```

For the various frame reception functions (GevWaitForNextFrame, GevGetNextFrame) the status of the image data should be checked by looking at the “status” member of the GEVBUF_HEADER to verify if all the data was received.

The actual image data received so far is present in the data buffer pointed to by "address" but the data may be incomplete if the “status” member is not 0.

Frame Status Values

Frame Status values returned by the *status* member are :

Define	Value	Definition
GEV_FRAME_STATUS_RECVD	0	Frame is complete.
GEV_FRAME_STATUS_PENDING	1	Frame is in progress. A frame (data) is currently being written to the buffer. This value is returned only in Asynchronous buffer cycling mode when the capture/receive thread is re-using this buffer internally for a subsequent frame at the same time as a previous frame is being examined by the program. In Synchronous buffer cycling mode, this value not returned as a buffer cannot be re-used internally until it is returned to the transfer for re-use.
GEV_FRAME_STATUS_TIMEOUT	2	Frame in-progress was not ready before timeout condition met. Acquisition of a frame (data) had started but was not completed before the specified frame timeout period expired. (The timeout period is the “streamFrame_timeout_ms“ member of the GEV_CAMERA_OPTIONS structure.) While there is data in the buffer from this frame, there is no way to know which data packets are missing. Possible reasons for the frame not being complete are: a) The timeout is set for too short of a time. This can happen with linescan camera having slow line rates or line triggers from an external source that can generate long frame times. b) Packets were dropped(*) and the timeout expired before all the resend operations were complete.

GEV_FRAME_STATUS_OVERFLOW	3	Frame in-progress was not complete before the internal queue of frames in-progress was full. The frame in-progress was abandoned before its completion due to a new frame arriving while the internal FIFO of frames in-progress was full. The internal FIFO size can be increased with the "*streamNumFramesBuffered*" member in the GEV_CAMERA_OPTIONS structure. While there is data in the buffer from this frame, there is no way to know which data packets are missing. Possible reasons for the internal FIFO of frames in-progress to be full are : a) The FIFO size is too small. b) The Frame rate from the camera is very high (kHz). When frame rates are higher than the operating system scheduler context switching rate (times the number of CPU cores) multiple frame completions can become pending simultaneously, causing the FIFO to fill while it waits to complete frames in order of their block ID. c) Packets were dropped(*) and the delay in the resend operations completing has caused the FIFO to fill while it waits to complete frames in order.
GEV_FRAME_STATUS_BANDWIDTH	4	Frame in-progress had too many resend operations . The frame in-progress had dropped packets(*) and too many packet resends have been generated in an attempt to recover the frame. The "*streamMaxPacketResends*" member in the GEV_CAMERA_OPTIONS structure controls the maximum number of retries. By default, it is set to twice the number of packets in a frame. While there is data in the buffer from this frame, there is no way to know which data packets are missing.
GEV_FRAME_STATUS_LOST	5	Frame in-progress had resend operations that failed. The frame in-progress had dropped packets(*) and at least one of the requests for a packet resend has failed. This frame has a missing packet that cannot be recovered so this frame is lost. While there is data in the buffer from this frame, there is no way to know which data packets are missing.
<other value>	<16-bit>	16-bit Status value from the camera itself. (Device / Vendor specific).

* To minimize the possibility of packets being dropped, the network tuning mechanisms provided in the gev_nettweak script should be adopted (see the Performance Tuning section). These are:

- Increasing the MTU minimizes the number of packets in a frame.
- Increasing the rx memory (net.core.rmem_max) allows more packets to queue on the network stack
- Increasing the packet backlog (net.core.netdev_max_backlog) allows more packets to queue before being processed onto the network stack.

For most Gigabit NIC types, using the PF_PACKET interface (via cap_net_raw or "sudo -E") with a maximized MTU provides the best defence against packet being dropped.

Supported Pixel Formats

The GigE Vision standard and the AIA's PFNC (Pixel Format Naming Convention) define a large set of pixel formats. Most formats can be handled by simply copying them to an application program's allocated buffer. Some format (namely packed formats), require processing to separate the pixels (or pixel components) from one another to aid in further processing or display.

The GigE-V Framework provides default internal processing for some formats that require it. The currently supported values for pixel format and the default processing available is provided in the following table. A blank entry for default handling, or a pixel format not specified here, indicates a simple copy of the data to the destination buffer.

Note: If "passthru_mode" is enabled for the connection to the camera, all formats are simply copied to the destination buffer, retaining their original payload layout.

Name (enum)	**Value**	**Description**	**Default Handling**
fmtMono8	0x01080001	8 Bit Mono Unsigned	
fmtMono8Signed	0x01080002	8 Bit Mono Signed	
fmtMono10	0x01100003	10 Bit Mono Unsigned	
fmtMono10Packed	0x010C0004	10 Bit Mono Packed	Unpacked to fmtMono10
fmtMono12	0x01100005	12 Bit Mono Unsigned	
fmtMono12Packed	0x010C0006	12 Bit Mono Packed	Unpacked to fmtMono12
fmtMono14	0x01100025	14 Bit Mono Unsigned	
fmtMono16	0x01100007	16 Bit Mono Unsigned	
fmtBayerGR8	0x01080008	8-bit Bayer GR	(*) See Note on Bayer support
fmtBayerRG8	0x01080009	8-bit Bayer RG	(*) See Note on Bayer support
fmtBayerGB8	0x0108000A	8-bit Bayer GB	(*) See Note on Bayer support
fmtBayerBG8	0x0108000B	8-bit Bayer BG	(*) See Note on Bayer support
fmtBayerGR10	0x0110000C	10-bit Bayer GR	(*) See Note on Bayer support
fmtBayerRG10	0x0110000D	10-bit Bayer RG	(*) See Note on Bayer support
fmtBayerGB10	0x0110000E	10-bit Bayer GB	(*) See Note on Bayer support
fmtBayerBG10	0x0110000F	10-bit Bayer BG	(*) See Note on Bayer support
fmtBayerGR10Packed	0x010C0026	10-bit Bayer GR packed	Unpacked to fmtBayerGR10
fmtBayerRG10Packed	0x010C0027	10-bit Bayer RG packed	Unpacked to fmtBayerRG10
fmtBayerGB10Packed	0x010C0028	10-bit Bayer GB packed	Unpacked to fmtBayerGB10
fmtBayerBG10Packed	0x010C0029	10-bit Bayer BG packed	Unpacked to fmtBayerBG10
fmtBayerGR12	0x01100010	12-bit Bayer GR	(*) See Note on Bayer support
fmtBayerRG12	0x01100011	12-bit Bayer RG	(*) See Note on Bayer support
fmtBayerGB12	0x01100012	12-bit Bayer GB	(*) See Note on Bayer support
fmtBayerBG12	0x01100013	12-bit Bayer BG	(*) See Note on Bayer support
fmtBayerGR12Packed	0x010C002A	12-bit Bayer GR packed	Unpacked to fmtBayerGR12
fmtBayerRG12Packed	0x010C002B	12-bit Bayer RG packed	Unpacked to fmtBayerRG12

fmtBayerGB12Packed	0x010C002C	12-bit Bayer GB packed	Unpacked to fmtBayerGB12
fmtBayerBG12Packed	0x010C002D	12-bit Bayer BG packed	Unpacked to fmtBayerBG12
fmtRGB8Packed	0x02180014	8 Bit RGB Unsigned	
fmtBGR8Packed	0x02180015	8 Bit BGR Unsigned	
fmtRGBA8Packed	0x02200016	8 Bit RGBA Unsigned	
fmtBGRA8Packed	0x02200017	8 Bit BGRA Unsigned	
fmtRGB10Packed	0x02300018	10 Bit RGB Unsigned	
fmtBGR10Packed	0x02300019	10 Bit BGR Unsigned	
fmtRGB12Packed	0x0230001A	12 Bit RGB Unsigned	
fmtBGR12Packed	0x0230001B	12 Bit BGR Unsigned	
fmtRGB14Packed	0x0230005E	14 Bit RGB Unsigned	
fmtBGR14Packed	0x0230004a	14 Bit BGR Unsigned	
fmtRGB16Packed	0x02300033	16 Bit RGB Unsigned	
fmtBGR16Packed	0x0230004B	16 Bit BGR Unsigned	
fmtRGBA16Packed	0x02400064	16 Bit RGBA Unsigned	
fmtBGRA16Packed	0x02400051	16 Bit BGRA Unsigned	
fmtYUV411packed	0x020C001E	YUV411 (composite color)	(*) See Note on Packed Color support
fmtYUV422packed	0x0210001F	YUV422 (composite color)	(*) See Note on Packed Color support
fmtYUV444packed	0x02180020	YUV444 (composite color)	(*) See Note on Packed Color support
fmtRGB10V1Packed	0x0220001C	10 Bit RGB custom V1	(*) See Note on Packed Color support
fmtRGB10V2Packed	0x0220001D	10 Bit RGB custom V2	(*) See Note on Packed Color support

Note : Bayer Support

By default, the GigE-V Framework will deliver Bayer formats to application programs as Monochrome data. A simple Bayer to RGB conversion capability is available in the common (shared) utility functions provided with the example programs. The Bayer conversion implementation provided is naïve in its complexity and is located outside of the Framework itself so it can be easily replaced by a more sophisticated converter of the end-user’s choice.

Note: Packed Color Support

Packed Color support is limited to conversion of the received pixel data to RGB for display purposes. Some older cameras from Teledyne DALSA (namely the Genie Color family) were able to output data in these formats. The original support for conversion to a displayable form remains in the common (shared) functions provided with the example programs.

Member Function Descriptions

The following functions are members of the Frame Acquisition group.

GevAbortTransfer

```
GEV_STATUS GevAbortTransfer(GEV_CAMERA_HANDLE handle);
```

Description

Stops the streaming transfer immediately.

Parameters

handle	Handle to the camera

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_XFER_NOT_INITIALIZED GEVLIB_ERROR_XFER_NOT_ACTIVE

GevFreeTransfer

```
GEV_STATUS GevFreeTransfer(GEV_CAMERA_HANDLE handle);
```

Description

Frees a streaming transfer to the list of buffers indicated.

Parameters

handle	Handle to the camera.

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_TIMEOUT (streaming thread did not respond within 5 seconds)

GevGetNextFrame

```
GEV_STATUS GevGetNextBuffer(GEV_CAMERA_HANDLE handle,
                            GEV_BUFFER_OBJECT **frame_object_ptr,
                            struct timeval *pTimeout);
```

Description

Waits for the next frame object to be acquired and returns its pointer. If no frame has been acquired before the timeout period expires, a NULL pointer is returned.

Parameters

handle	Handle to the camera
frame_object_ptr	Pointer to receive the frame object pointer.
pTimeout	Pointer to a struct timeval (microsecond precision) for the timeout period to wait for the next frame.

Return Value

GEV_STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_TIME_OUT
GEVLIB_ERROR_NULL_PTR

GevGetPayloadParameters

```
GEV_STATUS GevGetPayloadParameters(GEV_CAMERA_HANDLE handle, PUINT64 payload_size,
                                   PUINT32 format);
```

Description

Gets the values for the mandatory GigE Vision features “PayloadSize” and “PixelFormat” from the attached camera for the purposes of being able to determine the memory allocation requirements for any subsequent frame/data acquisition operations.

Parameters

payload_size	The returned size (in bytes) of the payload that will be sent from the device. This includes image data, metadata (chunks), etc.
format	The returned enumerated value for the payload format. The value depends on the camera/device model and mode of operation. See the “Supported Pixel Formats” section.

Return Value

GEV_STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_NULL_PTR
GEVLIB_ERROR_SOFTWARE

GevInitializeTransfer

```
GEV_STATUS GevInitializeTransfer(GEV_CAMERA_HANDLE handle, GevBufferCyclingMode
                                 mode, UINT64 bufSize, UINT32 numBuffers,
                                 UINT8 **bufAddress);
```

Description

Initializes a streaming transfer of frames to the list of buffers indicated. The size of the buffers and the buffer cycling mode is also set.

Parameters

handle	Handle to the camera.
mode	Buffer cycling mode. Can be either : **Asynchronous:** All buffers available all the time with no protection between the application and the acquisition process. Or **SynchronousNextEmpty;** Buffers obtained by the application are available only to the application until released back to the acquisition process. Buffers are filled in the order they are released back to the acquisition process. If there are no more buffers available to the acquisition process, subsequent images are not stored to memory and are deemed to have been sent to the "trash".
bufSize	The allocated size of buffers to be used in the transfer.
numBuffers	Number of buffers addresses in array.
bufAddress	Array of buffer addresses (already allocated).

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_PARAMETER_INVALID (GEV_REGISTER struct is not for an Integer register) GEVLIB_ERROR_ARG_INVALID (GEV_REGISTER definition is invalid) GEVLIB_ERROR_SOFTWARE (GEV_REGISTER struct defines an unsupported register type) Note: Errors include attempting to initialize the transfer on a connection that is not set up for streaming.

GevQueryTransferStatus

```
GEV_STATUS GevQueryTransferStatus(GEV_CAMERA_HANDLE handle,
                                  PUINT32 pTotalBuffers, PUINT32 pNumUsed,
                                  PUINT32 pNumFree, PUINT32 pNumTrashed,
                                  GevBufferCyclingMode *pMode);
```

Description

This function returns status information about the frame transfer currently in progress. The total number of buffers associated with the transfer are returned alongwith the number of filled buffers, the number of free buffers available, and the number of buffers sent to trash The buffer cycling mode is also returned. If the buffer cycling mode is set to Synchronous, any frames from the camera that arrive when no free buffers are available are sent to trash (not stored) and the number of trashed buffers is incremented. This information can be used to tell if the application is falling behind in its handling of frames from the camera.

Parameters

handle	Handle to the camera
pTotalBuffers	Pointer to receive the total number of buffers in the transfer list.
pNumUsed	Pointer to receive the number of filled buffers ready to be received from the transfer list.
pNumFree	Pointer to receive the number of empty (free) buffers that are available to be filled.
pNumTrashed	Pointer to receive the total number of buffers that have been “trashed” so far. (i.e. Frames that are dropped when there are no more empty buffers to fill but image data has still been received).
pMode	Pointer to receive the current buffer cycling mode (Asynchronous=0, SynchronousNextEmpty=1).

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_PARAMETER_INVALID GEVLIB_ERROR_ARG_INVALID

GevReleaseFrame

```
GEV_STATUS GevReleaseFrame(GEV_CAMERA_HANDLE handle,
                           GEV_BUFFER_OBJECT **frame_object_ptr);
```

Description

Releases a buffer object back to the acquisition process for re-use. It is mandatory to call this function for a transfer using the SynchronousNextEmpty cycle mode in order to avoid running out of buffers for the acquisitions process to fill. It is not necessary to call this function for a transfer using the Asynchronous cycle mode.

Parameters

handle	Handle to the camera
frame_object_ptr	Pointer to the buffer object being released.

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_PARAMETER_INVALID GEVLIB_ERROR_ARG_INVALID

GevReleaseFrameBuffer

```
GEV_STATUS GevReleaseFrameBuffer(GEV_CAMERA_HANDLE handle, void **frame_buffer_ptr);
```

Description

Releases a buffer object back to the acquisition process for re-use. The buffer object is identified from the frame buffer pointer passed in to the function. It is mandatory to call this function for a transfer using the SynchronousNextEmpty cycle mode in order to avoid running out of buffers for the acquisition process to fill. It is not necessary to call this function for a transfer using the Asynchronous cycle mode.

Parameters

handle	Handle to the camera
frame_buffer_ptr	Pointer to the frame buffer data for the image object being released,.

Return Value

GEV_STATUS Possible values are

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_PARAMETER_INVALID
GEVLIB_ERROR_ARG_INVALID

GevStartTransfer

```
GEV_STATUS GevStartTransfer(GEV_CAMERA_HANDLE handle, UINT32 numFrames);
```

Description

Starts the streaming transfer.

Parameters

handle	Handle to the camera
numFrames	Number of frames to be acquired (-1 for continuous).

Return Value

GEV_STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEV_STATUS_BUSY (camera is busy reconfiguring – try again later)
GEVLIB_ERROR_XFER_NOT_INITIALIZED
GEVLIB_ERROR_XFER_ACTIVE

GevStopTransfer

```
GEV_STATUS GevStopTransfer(GEV_CAMERA_HANDLE handle);
```

Description

Stops the streaming transfer.

Parameters

handle	Handle to the camera

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_XFER_NOT_INITIALIZED GEVLIB_ERROR_XFER_NOT_ACTIVE

GevWaitForNextFrame

```
GEV_STATUS GevWaitForNextFrame(GEV_CAMERA_HANDLE handle,
                               GEV_BUFFER_OBJECT **frame_object_ptr,
                               struct timeval *pTimeout);
```

Description

Waits for the next frame object to be acquired and returns its pointer. If no frame has been acquired before the timeout period expires, a NULL pointer is returned.

Parameters

handle	Handle to the camera
frame_object_ptr	Pointer to receive the frame object pointer.
timeout	Timeout period (in msec) to wait for the next frame to arrive.

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_TIME_OUT GEVLIB_ERROR_NULL_PTR

Asynchronous Camera Event Handling

The GVCP asynchronous message channel is available only to applications using the primary control channel. Support for it is automatically enabled when a camera is opened with access mode GevExclusiveMode or GevControlMode.

The supported EVENT_CMD and EVENTDATA_CMD events are found in the device's XML file. Signaling of these events needs to be enabled via calls to GevWriteReg using the proper address and enable values.

GigE-V Framework API allows an application to register two actions for an event. On receipt of an event, an application may have a callback function invoked and/or an event object can be signaled. In this case the application event is signaled before the callback function is invoked. A single call to GevUnregisterEvent will cause both the application event and the callback function to be unregistered.

Note that the callback is performed synchronously with the delivery of the event message from the camera. Care should be taken to complete the callback processing quickly so that subsequent messages are not lost. If lengthy processing is required, the callback is responsible for saving the contents of the EVENT_MSG data structure and the "data" buffer and signaling some other asynchronous processing context (thread) to perform that processing. Once the callback function returns, the contents of the EVENT_MSG structure (msg) and the 'data' buffer are no longer valid and will be overwritten by the asynchronous message.

The following functions provide this service.

Member Function Overview

Function	Description
GEVEVENT_CBFUNCTION	Type Definition
GevRegisterEventCallback	Register an Event Callback
GevRegisterApplicationEvent	Register an Application Event
GevUnregisterEvent	Un-register an Application Event

Member Function Descriptions

The following functions are members of the Asynchronous Camera Event Handling group.

GEVEVENT_CBFUNCTION

```
typedef void (*GEVEVENT_CBFUNCTION)
                    (PEVENT_MSG msg, PUINT8 data, UINT16 size, void *context);
```

Parameters

msg Pointer to an EVENT_MSG structure containing information on the intercepted event. Here the data structure is defined as :

```
typedef struct
{
    UINT16 eventNumber;
    UINT16 streamChannelIndex;
    UINT64 blockId;
    UINT64 timestamp;
    UINT32 timeStampHigh;
    UINT32 timeStampLow;
} EVENT_MSG, *PEVENT_MSG;
```

where:

eventNumber The event number that caused the callback to be invoked.

*streamChannelIndex*The streaming data channel identifier that caused the event to be sent in the first place.

blockId The blockId associated with this event.

timestamp 64-bit timestamp for this event (based on camera's timestamp timebase).

timeStampHigh High (MSB) 32-bits of 64-bit timestamp

timeStampLow Low (LSB) 32-bits of 64-bit timestamp

data Pointer to event data returned from the camera if the particular event intercepted also sends data. It is NULL if not data has been sent.

size Size of the event data returned by the camera.
(It is zero if the particular event intercepted does not send any data).

context Pointer to context data set up at the time of the callback's registration.

Return Value

VOID

GevRegisterApplicationEvent

```
GEV_STATUS GevRegisterApplicationEvent(GEV_CAMERA_HANDLE handle,
                                       UINT32 EventID, _EVENT appEvent);
```

Description

Registers an Application Event

Parameters

handle GEV_CAMERA_HANDLE identifying the camera whose events are to be intercepted by the application.

EventID Specific EventID of the event to be intercepted. They are usually defined in the XML file for the camera.

appEvent Event handle. The _EVENT type is aliased to the HANDLE data type used by the CorW32 helper library that provides WIN32-like constructs to the Linux environment. In this case, the HANDLE is for a WIN32-like event that is, essentially, a thin wrapper around a pthread condition variable.

Return Value

GEV_STATUS GEVLIB Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEV_STATUS_ERROR (too many registration calls have been made for this camera – 16 maximum)

GevRegisterEventCallback

```
GEV_STATUS GevRegisterEventCallback(GEV_CAMERA_HANDLE handle,  UINT32 EventID,
                                    GEVEVENT_CBFUNCTION func, void *context);
```

Description

Registers an Event Callback

Parameters

handle GEV_CAMERA_HANDLE identifying the camera whose events are to be intercepted by the application.

EventID Specific EventID of the event to be intercepted. They are usually defined in the XML file for the camera.

func Function to call when EventID is signaled. The function is of type GEVEVENT_CBFUNCTION.

context Pointer to context data set up at the time of the callback's registration and passed to 'func'.

Return Value

GEV_STATUS GEVLIB Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEV_STATUS_ERROR (too many registration calls have been made for this camera – 16 maximum)

GevUnregisterEvent

```
GEV_STATUS GevUnregisterEvent(GEV_CAMERA_HANDLE handle, UINT32 EventID);
```

Description

Un-register an Application Event

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera whose events are to be intercepted by the application.
EventID	The particular EventID of the event to be intercepted. They are usually defined in the XML file for the camera.

Return Value

GEV_STATUS GEVLIB	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEV_STATUS_ERROR (eventID not found)

Manual Camera Detection and Configuration (Advanced Topic)

For situations where the automatic detection and configuration of cameras is not wanted, functions are provided to manually set up the camera in the system.

Member Function Overview

Function	Description
GevEnumerateNetworkInterfaces	Fills a list of network interfaces visible from the application.
GevEnumerateGevDevices	Fills a list of device interfaces visible from the application through a particular network interface.
GevSetCameraList	Manually fills the internal camera information list.
GevForceCameraIPAddress	Forces the IP address of a device to a known value.
Gev_Reconnect	Reconnects a camera that has become disconnected.

Structure Definition: GEV_NETWORK_INTERFACE

```
typedef struct
{
    BOOL fIPv6;
    UINT32 ipAddr;
    UINT32 ipAddrLow;
    UINT32 ipAddrHigh;
    UINT32 ifIndex;
} GEV_NETWORK_INTERFACE, *PGEV_NETWORK_INTERFACE;
```

Where:

- fIPv6 Is TRUE/FALSE for the NIC having an IPv6 address.
  (GigE Vision is currently only supported on IPv4).
- ipAdd 32-bit IP address (IPv4) for the NIC card.
- ipAddrLow Low 32-bits of a 64-bit IPv6 address for the NIC card.
  (GigE Vision is currently only supported on IPv4).
- ipAddrHigh High 32-bits of a 64-bit IPv6 address for the NIC card.
  (GigE Vision is currently only supported on IPv4).
- ifIndex The O/S internal index of the network interface, set by the system.
  It is required for the GigE-V Framework API under Linux to provide access to the high performance packet interface (PF_PACKET protocol).

Structure Definition: GEV_CAMERA_INFO

```
typedef struct
{
    BOOL fIPv6;
    UINT32 ipAddr;
    UINT32 ipAddrLow;
    UINT32 ipAddrHigh;
    UINT32 macLow;
    UINT32 macHigh;
    GEV_NETWORK_INTERFACE host;
    UINT32 capabilities;
    char   manufacturer[65];
    char   model[65];
    char   serial[65];
    char   version[65];
    char   username[65];
} GEV_CAMERA_INFO, *PGEV_CAMERA_INFO;
```

Member Function Descriptions

The following functions are members of the Manual Camera Detection and Configuration (Advanced Topic) group.

Gev_Reconnect

```
GEV_STATUS Gev_Reconnect(GEV_CAMERA_HANDLE handle);
```

Description

Reconnects a camera that has become disconnected. A camera can become disconnected when it is temporarily/briefly unplugged from the network. A disconnected camera cannot always be restored using this function. If an error is returned, the program should consider closing and re-opening the camera and restarting any initialized transfers.

Note: A disconnection that results in the camera losing its IP address cannot be recovered from. A camera can lose its IP address through a power cycle, through having the camera’s heartbeat timer expire (usually due to running an application in a debugger and remaining too long at a breakpoint), or through unplugging the network cable when the camera is not in a persistent IP address mode.

Parameters

handle	Camera handle

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEV_STATUS_ERROR (camera is not actually disconnected) Other error from writing to the camera.

GevEnumerateGevDevices

```
GEV_STATUS GevEnumerateGevDevices(GEV_NETWORK_INTERFACE *pIPAddr,
                                  UINT32 discoveryTimeout,
                                  GEV_DEVICE_INTERFACE *pDevice, UINT32 maxDevices,
                                  PUINT32 pNumDevices);
```

Description

Fills a list of device interfaces visible from the application through a particular network interface.

Parameters

pIPAddr	Pointer to the GEV_NETWORK_INTERFACE structure to use to query the attached network for the presence of GigE Vision camera devices.
discoveryTimeout	Time, in milliseconds, to wait for a response from cameras on the attached network.
pDevice	Pointer to an array of GEV_DEVICE_INTERFACE (also known as GEV_CAMERA_INFO) structures to contain information for cameras found on the attached network.
maxDevices	Maximum number of entries in the list pointed to by pDevice.
pNumDevices	Pointer to contain the number of devices found on the network.

Return Value

GEV_STATUS	Possible values are: GEV_STATUS_SUCCESS GEV_STATUS_ERROR (an internal error in the library)

GevEnumerateNetworkInterfaces

```
GEV_STATUS GevEnumerateNetworkInterfaces (GEV_NETWORK_INTERFACE *pIPAddr,
                                          UINT32 maxInterfaces,
                                          PUINT32 pNumInterfaces);
```

Description

Fills a list of network interfaces visible from the application.

Parameters

pIPAddr	Network interface data structure (GEV_NETWORK_INTERFACE) to contain information found for NIC cards in the system.
maxInterfaces	Maximum number of interfaces for which there is storage in pIPAddr.
pNumIntefaces	Number of network interfaces found.

Return Value

GEV_STATUS	Always returns success (GEV_STATUS_SUCCESS / GEVLIB_OK)

GevForceCameraIPAddress

```
GEV_STATUS GevForceCameraIPAddress(UINT32 macHi, UINT32 macLo, UINT32 IPAddress,
                                   UINT32 subnetmask);
```

Description

Forces the IP address of a device to a known value. It allows for recovery from incorrect IP address configuration. The device is identified by its MAC address and uses the known network interface list (stored internally) to locate and access the camera for reconfiguration.

Parameters

macHi	Hi 16 bits of the 48 bit MAC address for device.
macLo	Low 32 bits of the 48 bit MAC address for device.
ip	IP address to assign to the device when it is fond. (IPv4).
subnetmask	Subnet mask to be assigned to the camera when it is found.

Return Value

GEV_STATUS	Possible values are: GEV_STATUS_SUCCESS GEV_STATUS_ERROR NOTE: A returned error may indicate that the command was silently discarded rather than being an actual error.

GevSetCameraList

```
GEV_STATUS GevSetCameraList(GEV_CAMERA_INFO *cameras, int numCameras);
```

Description

Manually fills the internal camera list containing information on the GigE Vision device of interest to the API. This allows an application to manually set up only the cameras it is interested in and skip the "automatic" detection step.

Note: If the camera list is set manually (with at least one camera), all calls to the GevGetCameraList function will return this manually set list. No further automatic detection will be performed. Automatic detection can be re-enabled by setting a zero length (NULL) camera list with this function.

Parameters

camera	Pointer to a list of GEV_CAMERA_INFO entries.
numCameras	Number of camera / device entries in the list

Return Value

GEV_STATUS	Only returns GEVLIB_OK

Utility Functions

The following functions are provided as useful utility functions for manipulating image formats used to define image buffer storage.

GevGetBayerAsRGBPixelType

```
UINT32 GevGetBayerAsRGBPixelType(UINT32 pixelType);
```

Description

Returns a color (RGB) pixel type corresponding to the output of a simple Bayer to color conversion function. Color component bit depth are preserved. It is intended to assist in allocating space for a converted image and to provide an output format for the Bayer conversion function to use.

(If the input pixel type is not recognized as Bayer, then fmtRGBA8Packed will be returned.)

Parameters

pixelType	GigE Vision pixel data format (packed).

Return Value

UINT32	The RGB pixel format corresponding to the input format when it is converted. (e.g. fmtBayerBG12 becomes fmtRGB12Packed)

GevGetConvertedPixelType

```
UINT32 GevGetConvertedPixelType(int convertBayer, UINT32 pixelType);
```

Description

Returns the converted pixel type that will result from the default processing applied during the acquisition process. If convertBayer is 1 (True), then the output converted pixel type for both Bayer and PackedBayer formats will be a corresponding RGB format. If convertBayer is 0 (False), then the output converted pixel type for both Bayer and PackedBayer formats will be the Bayer format itself. Packed Monochrome formats will output their unpacked equivalent.

(This is a combination of the previous functions GevGetUnpackedPixelType and GevGetBayerAsRGBPixelType)

Parameters

pixelType	GigE Vision pixel data format (packed).

Return Value

UINT32	The converted pixel type that the acquisition process will supply. (e.g. fmtBayerBG12Packed becomes fmtBayerBG12)

GevGetPixelComponentCount

```
UINT32 GevGetPixelComponetCount(UINT32 pixelType);
```

Description

Returns the number of color components in a pixel for the input raw (GigE Vision) image format. It is intended for simplifying display and storage functions. (Monochrome images have a single color component).

Parameters

pixelType	GigE Vision pixel data format.

Return Value

UINT32	The number of components in a pixel.

GevGetPixelDepthInBits

```
UINT32 GevGetPixelDepthInBits(UINT32 pixelType);
```

Description

Returns the number of bits taken up by a single color component in a pixel for the input raw (GigE Vision) image format. It is intended for simplifying display and LUT functions.

Note: YUV composite color pixel formats need to be converted to an RGB equivalent. The various Y/U/V packed combinations may be (incorrectly) treated as 8 bit data.

Parameters

pixelType	GigE Vision pixel data format.

Return Value

UINT32	The depth of the pixel in bits

GevGetPixelSizeInBytes

```
UINT32 GevGetPixelSizeInBytes(UINT32 pixelType);
```

Description

Returns the number of bytes taken up by the input raw (GigE Vision) image format.

Parameters

pixelType	GigE Vision pixel data format.

Return Value

UINT32	Size of the pixel in bytes

GevGetUnpackedPixelType

```
UINT32 GevGetUnpackedPixelType(UINT32 pixelType);
```

Description

Returns the pixel format that would result following a simple unpacking of a packed input pixel format by the default processing during frame acquisition.

(If the input pixel type is not packed, it is returned as the unpacked format).

Parameters

pixelType	GigE Vision pixel data format (packed).

Return Value

UINT32	The pixel format corresponding to the input format when it is unpacked. (e.g. fmtMono12Packed becomes fmtMono12)

GevIsPixelTypeMono, GevIsPixelTypeRGB, GevIsPixelTypePacked, GevIsPixelTypeBayer

```
BOOL GevIsPixelTypeMono(UINT32 pixelType);
BOOL GevIsPixelTypeRGB(UINT32 pixelType);
BOOL GevIsPixelTypePacked(UINT32 pixelType);
BOOL GevIsPixelTypeBayer(UINT32 pixelType);
```

Description

Returns true/false for the various image pixel types (mono, RGB, packed, Bayer).

Parameters

pixelType GigE Vision pixel data format.

Return Value

BOOL True/False (for the condition queried).

GevTranslateRawPixelFormat

```
GEV_STATUS GevTranslateRawPixelFormat(UINT32 rawFormat, PUINT32 translatedFormat,
                                      PUINT32 bitDepth, PUINT32 order)
```

Description

Translates an input raw (GigE Vision) image format into information useful during image display.

Parameters

rawFormat GigE Vision pixel data format.

translatedFormat Simplified version of the format. Possible values are:
GEV_PIXEL_FORMAT_MONO, GEV_PIXEL_FORMAT_MONO_PACKED,
GEV_PIXEL_FORMAT_RGB, GEV_PIXEL_FORMAT_RGB_PACKED,
GEV_PIXEL_FORMAT_BAYER, GEV_PIXEL_FORMAT_YUV,
GEV_PIXEL_FORMAT_RGB_PLANAR

bitDepth Number of bits in a mono pixel or in each color channel..

order Color channel order. Possible values are:
GEV_PIXEL_ORDER_NONE (for MONO and YUV)
GEV_PIXEL_ORDER_RGB,
GEV_PIXEL_ORDER_BGR,
GEV_PIXEL_ORDER_GRB,,
GEV_PIXEL_ORDER_GBR,
GEV_PIXEL_ORDER_RGB10V1 (a custom 10-bit RGB)
GEV_PIXEL_ORDER_RGB10V2 (a custom 10-bit RGB)

Return Value

BOOL True/False (for the condition queried).

Operating System Independence Wrapper

The OS Independence wrapper provides a compatibility layer allowing GigE-V Framework API to be (potentially) used in multiple operating system environments. It uses functions from the WIN32 compatibility library (libCorW32) provided with the installation.

Function Overview

Function	Description

BOOL _CreateEvent (_EVENT *pEvent);
BOOL _DestroyEvent (_EVENT *pEvent);
BOOL _WaitForEvent (_EVENT *pEvent, UINT32 *timeout*);
BOOL _ClearEvent (_EVENT *pEvent);
BOOL _SetEvent (_EVENT *pEvent);
	Event objects:
	Required functions for manual reset event signaling

BOOL _InitCriticalSection (_CRITICAL_SECTION *pCSection);
BOOL _ReleaseCriticalSection (_CRITICAL_SECTION *pCSection);
BOOL _EnterCriticalSection (_CRITICAL_SECTION *pCSection);
BOOL _LeaveCriticalSection (_CRITICAL_SECTION *pCSection);
	Critical Section objects required functions

BOOL _CreateThread (unsigned _stdcall fct(void *), void *context, int priority, _THREAD *pThread);
BOOL _WaitForThread (_THREAD *pThread, UINT32 *timeout*);
	Thread objects required functions:

Appendix A: Common Package Management methods in Linux

As part of installing the GigE-V Framework for Linux, other software packages are either useful or required for proper functioning of the API. Software packages are available for distribution in different file formats. The most common ones are:

- ".deb" files: Debian package files
- ".rpm" files: RedHat Package Manger files
- ".tgz": Compressed tar archive files

Different Linux distributions use different programs for managing (searching, installing, updating) these packages. Distributions usually have both a graphical program used for installing packages as well as a command-line program for installing packages.

Software Package Management Tools

Linux Distribution (Family)	GUI-based Tool	Command Line Tool
Ubuntu	Ubuntu Software Center	apt
Debian	Synpatic (among others)	apt
Suse/openSuse	Yast	zypper
Red Hat (RHEL/Fedora/CentOS/Scientific)	"Add / Remove Software" menu item gnome-packagekit-installer yumex	yum (for older releases) dnf (for recent releases)
Other	See distro documentation	See distro documentation

The common tasks provided by package managers are :

1) Managing (Install/Remove) Packages

   This is the most frequently used set of tasks performed by a package manager. The functions include :

   - Installing package from a repository
   - Installing package from a file obtained elsewhere than a repository
   - Updating an installed package
   - Uninstalling a package.

2) Searching for Packages

   The known repositories can be searched for packages by name. Descriptive information about the packages can be displayed and the list of packages actually installed can be found.

3) Updating Package Repository Information

   Each distribution has its own default list of repositories plus lists of extra repositories that can be added (by URL) should they be required in order to locate a package. Updating the repository information involves the following functions :

   - Updating package lists with the latest information
   - Listing known repositories
   - Adding repositories to the known list
   - Removing repositories from the known list

CLI Package Management Command Examples (by Distribution)

The following is a summary of the commands (and options) that can be used on some, more popular, Linux distributions for finding and installing the packages used by the GigE-V Framework for Linux.

Task	**apt (.deb)** (Ubuntu/Debian family)	**yum (.rpm)** (older RedHat family)	**dnf (.rpm)** (newer RedHat family)	**zipper (.rpm)** (Suse/openSuse family)
Update package list	apt-get update	yum check-update	dnf check-update	zypper refresh
install from repository	apt-get install pkgname	yum install pkgname	dnf install pkgname	zypper install pkgname
update installed package	apt-get install pkgname	yum update pkgname	dnf update pkgname	zypper update –t package pkgname
remove package	apt-get remove pkgname	yum erase pkgname	dnf erase pkgname	zypper remove pkgname
show package info	apt-cache show pkgname	yum info pkgname	dnf info pkgname	zypper info pkgname
list installed packages	dpkg -l	rpm -qa	rpm -qa	zypper search -is
search for package by name : by pattern :	apt-cache search pkgname apt-cache search pattern	yum list pkgname yum search pattern	dnf list pkgname dnf search pattern	zypper search pkgname zypper search –t pattern pattern
list known repos	cat /etc/apt/sources.list	yum reposlist	dnf repolist	zypper repos
add repository	Add URL to file /etc/apt/sources.list	Add *.repo files to /etc/yum.repos.d	Add *.repo files to /etc/yum.repos.d And/or edit /etc/dnf/dnf.conf	zypper addrepo URL reponame
remove repository	Remove URL from file /etc/apt/sources.list	Remove *.repo files from /etc/yum.repos.d	Remove *.repo files from /etc/yum.repos.d And/or edit /etc/dnf/dnf.conf	zypper removerepo reponame

Usually, if the command line program cannot find the desired package, the graphical program can be used to search using regular expression patterns to find candidates and the package information / descriptions returned can be used to determine which package to install.
Note: Different Linux distributions sometimes call the same packages by different, but similar, names. Some attention is required in order to ensure that the proper package is found and installed.

Required Packages

The following table contains a list of packages needed. In some cases the names are different or need to be searched for using a pattern due to distribution-dependent naming conventions.

Purpose	Distribution	Package Name
S/W Development (Compilers/Linkers etc….)	Ubuntu / Debian	gcc (top level package for C compiler) and g++ (top level package for C++ compilation)
	Suse/openSuse	gcc gcc-c++
	Fedora/RHEL/CentOs	gcc gcc-c++
Packet capture (for PF_PACKET interface support)	Ubuntu/Debian	libpcap0.8
	Suse/openSuse	libpcap1
	Fedora/RHEL/CentOs	Search for libcap*
Load ".glade" UI definition files at application runtime	Ubuntu/Debian	libglade2-0 libglade2-dev
	Suse/openSuse	libglade-2_0-0 libglade2-devel
	Fedora/RHEL/CentOs	Search for libglade2*
Compile and Link Demos using X11 for Image display	Ubuntu/Debian	libx11-dev libxext-dev
	Suse/openSuse	xorg-x11-libX11-devel xorg-x11-libXext
	Fedora/RHEL/CentOs	Search for libXext* Search for libX11-devel (may need rpmfind for this).
Capabilities setting for CAP_NET_RAW and CAP_SYS_NICE support	Ubuntu / Debian	libcap2 or libcap-ng0
	Suse/openSuse	libcap2 or libcap-ng0 and libcap-progs
	Fedora/RHEL/CentOs	Search for libcap*
Compile and link GigE Vision Device Status tool	Ubuntu / Debian	libgtk-3-dev
	Suse/openSuse	gtk2-devel
	Fedora/RHEL/CentOs	gtk2-devel
TIFF file operations	Ubuntu / Debian	libtiff-dev
	Suse/openSuse	libtiff-devel
	Fedora/RHEL/CentOs	libtiff-devel

Appendix B: Helper Functions

The following functions are provided in a common (shared) directory (in $HOME/DALSA/GigeV/examples/common) for use by example programs and are available for end-user applications. They are provided as "helper" functions and are used for interacting with Linux-specific aspects of the system; as such they are not part of the GigE-Vision Framework API itself.

IsGevPixelTypeX11Displayable

```
int IsGevPixelTypeX11Displayable(UINT32 pixelType);
```

Description

Returns true/false (1/0) if the input GigE Vision pixel type is displayable by the X11 Utility function provided with the example programs.

Parameters

pixelType	GigE Vision pixel data format.

Return Value

int	True/False (1/0) that X11 display functions support the specified format.

GetX11DisplayablePixelFormat

```
UNIT32 GetX11DisplayablePixelFormat (int convertBayer, UINT32 rawGevPixelFormat,
                                     UINT32 *convertedGevPixelFormat,
                                     UINT32 displayableSaperaPixelFormat);
```

Description

Returns the converted pixel type that allows the input GigE Vision pixel type to be displayed after a conversion has been applied to it. If convertBayer is 1 (True), any Bayer input formats return a color pixel format.

Currently, all Bayer formats converted for display use format *fmtBayerBGRA8* due to a legacy implementation in the display helper functions.

Parameters

rawGevPixelFormat	Input GigE Vision pixel data format.
convertedGevPixelFormat	Returned GigE Vision pixel format reflecting any conversion performed during frame acquisition (e.g. unpacking, etc…).
displayableSaperaPixelFormat	Returned pixel format displayable by the X11 utilities shared with our SaperaLT API.

Return Value

UINT32	Always 0.

CreateDisplayWindow

```
X_VIEW_HANDLE CreateDisplayWindow(const char *title, int visible, int height,
                                  int width, int depth, int sapera_format,
                                  int use_shared_memory);
```

Description

Creates an X11 display window.

Parameters

title	Window title
visible	
height	Window height, in pixels
width	Window width, in pixels
depth	Pixel depth, in bits
use_shared_memory	

Return Value

X_VIEW_HANDLE	Handle to window

DestroyDisplayWindow

```
void DestroyDisplayWindow (X_VIEW_HANDLE xhandle);
```

Description

Destroys all resources allocated to an X11 display window.

Parameters

xhandle	Handle to window

Return Value

void

Display_Image

```
int Display_Image(X_VIEW_HANDLE xhandle, int depth, int width, int height,
                  void *image);
```

Description

Creates an X11 display window.

Parameters

xhandle	Handle to window
depth	Pixel depth, in bits
width	Window width, in pixels
height	Window height, in pixels
void	pointer to image

Return Value

int	Returns non-zero value on success.

ConvertGevImageToX11Format

```
void ConvertGevImageToX11Format (int w, int h, int gev_depth, int gev_format,
                                 void *gev_input_data, int x11_depth,
                                 int x11_format, void *x11_output_data);
```

Description

Creates an X11 display window.

Parameters

w	Window width, in pixels
h	Window height, in pixels
depth	Gev image pixel depth, in bits
gev_format	Gev image format. Possible values are: fmtBayerBG10Packed fmtBayerGB10Packed fmtBayerGR10Packed fmt
**gev_input_data*	Pointer to gev image data
x11_depth	X11 image pixel depth, in bits
x11_format	X11 image format
**x11_output_data*	Pointer to memory location for X11 output image

Return Value

int	Returns non-zero value on success.

Read_TIFF_ToGevImage

```
int Read_TIFF_ToGevImage(char *filename, uint32_t *width, uint32_t *height,
                         int pixel_format, int size, void *imageData);
```

Description

Reads a TIFF image from file and converts it to a GevImage. Note, reading 10/12/14/16-bit formats into 8-bit formats is not supported.

Parameters

**filename*	Name of TIFF file
**width*	Pointer to memory to hold width, in pixels, of TIFF file image to read
**height*	Pointer to memory to hold height, in pixels, of TIFF file image to read
pixel_format	Output image pixel format
size	Size of buffer to hold image read, in bytes
**imageData*	Pointer to memory to hold output GevImage

Return Value

int	Returns non-zero value on success. Possible error values include: GEVLIB_ERRPR_NULL_PTR: Data pointer is NULL. GEVLIB_ERROR_INVALID_PIXEL_FORMAT: The specified pixel format is not supported.

Write_GevImage_ToTIFF

```
int Write_GevImage_ToTIFF (char *filename, uint32_t width, uint32_t height, int
                           pixel_format, void *imageData);
```

Description

Writes the input image to the specified TIFF file.

Parameters

**filename*	Name of file to write
width	Width, in pixels, of image to write.
height	Height, in pixels, Pointer to memory to hold height of TIFF file image to read
pixel_format	Image data pixel format
**imageData*	Pointer to memory to hold output TIFF image

Return Value

int	When successful, returns the number of bytes written to the file. Possible error values include: GEVLIB_ERROR_NULL_PTR: Data pointer is NULL. GEVLIB_ERROR_INVALID_PIXEL_FORMAT: The specified pixel format is not supported.

ConvertBayerToRGB

```
GEV_STATUS ConvertBayerToRGB ( int convAlgorithm, UINT32 h, UINT32 w, UINT32
                               inFormat, void *inImage, UINT32 outFormat, void
                               *outImage);
```

Description

Converts a Bayer image to an RGB image. Supported conversions are:

- 8-bit Bayer to 8-bit RGB
- 16-bit Bayer to 10/12/14/16-bit RGB
- 16-bit Bayer to 8-bit RGB (typically for display purposes)

Parameters

convAlgorithm	Conversion algorithm. Currently, only BAYER_CONVERSION_2X2 is supported.
h	Height, in pixels, of Bayer image to convert.
w	Width, in pixels, of Bayer image to convert.
inFormat	Bayer format of image to convert. Refer to
outFormat	Output image RGB format
**outImage*	Pointer to memory to hold output RGB image

Return Value

Possible return values include:
GEVLIB_ERROR_PARAMETER_INVALID: Unsupported input format or algorithm.

Appendix C: Feature Access Through Static Registers

A set of functions is provided to directly access camera registers. Standard features are implemented as simple registers using a static device-specific table of GEV_REGISTER structure definitions.

Note : These function operate outside of the GenICam XML based feature access functions (see above) and require manual configuration of the static register table in order to work. They remain in the API for support of legacy applications, legacy cameras and memory constrained embedded environments.

Member Function Overview

Function	Description
GevGetCameraRegisters	Get the Camera Registers
GevSetCameraRegInfo	Set the Camera Register Info
GevInitCameraRegisters	Initialize Camera Registers
GevGetNumberOfRegisters	Get the number of Camera register entries configured for the camera
GevReadRegisterByName	Read the contents of a Camera Register by name.
GevWriteRegisterByName	Write the contents of a Camera Register byname.
GevGetRegisterNameByIndex	Get the name of a Camera register entry based on its index
GevGetRegisterByName	Get a Camera Register structure by name
GevGetRegisterPtrByName	Get a Pointer to a Camera Register structure by name
GevGetRegisterByIndex	Get a Camera Register structure by index
GevGetRegisterPtrByIndex	Get a Pointer to a Camera Register structure by index.
GevRegisterRead	Read Register (a generic register access function)
GevRegisterWrite	Write Register (a generic register access function)
GevRegisterWriteNoWait	Write Register without waiting for an ack (a generic register access function)
GevRegisterWriteArray	Write multiple values to a memory area.
GevRegisterReadArray	Read multiple values from a memory area.
GevRegisterWriteInt	Write an integer to a register (an integer register access function)
GevRegisterReadInt	Read an integer from a register (an integer register access function)
GevRegisterWriteFloat	Write a float to a register (a float register access function)
GevRegisterReadFloat	Read a float from a register (a float register access function)

Member Function Descriptions

The following functions are members of the Camera Register / Feature Access group. They operate on the GEV_REGISTER data structure.

For informational purposed, this data structure is defined as:

```
typedef struct
{
    char featureName[FEATURE_NAME_MAX_SIZE];     // String name of feature for this register.
    UINT32 address;                     // Address for accessing feature in camera
                                        // NOREF_ADDR if not in camera).
    RegAccess accessMode;               // RO, WO, RW access allowed.
    BOOL32 available;                   // True if feature is available (in camera or not)
                                        // False is not available.
    RegType type;                       // String, Float, Integer, Enum, Bit, Area, Fixed …
    UINT32 regSize;                     // Size of storage for register
                                        // (or register set / area).
    UINT32 regStride;                   // Increment between register items accessed via selector
    UINT32 minSelector;                 // Minimum value for selector
                                        // (corresponds to base address).
    UINT32 maxSelector;                 // Maximum value for selector.
    GENIREG_VALUE. value;               // Current value
                                        // (storage for features not backed by a register).
    GENIREG_VALUE minValue;             // Minimum allowable value.
    GENIREG_VALUE maxValue;             // Maximum allowable value.
    UINT32 readMask;                    // AND Mask for read (integers only)
    UINT32 writeMask;                   // AND Mask for write (integers only)
    PGENICAM_FEATURE feature;           // Pointer to feature in feature table (future).
    char selectorName[FEATURE_NAME_MAX_SIZE];    // String name of register-based selector
                                                 // for feature.
    char indexName[FEATURE_NAME_MAX_SIZE];       // String name of index (second selector)
                                                 // for feature.
} GEV_REGISTER, *PGEV_REGISTER;
```

Some functions operate on the DALSA_GENICAM_GIGE_REGS data structure (refer to the *gevapi.h* file in the DALSA/GigeV/include directory) which is a set of GEV_REGISTER structures organized along the lines of the GenICam Standard Features Naming Convention (SFNC) version 1.2.1. The SFNC documentation is available at http://www.emva.org/standards-technology/genicam/.

Note: The GEV_REGISTER structure and its access methods are a work-in-progress. While the functions in the API are expected to remain the same, the underlying setup of the GEV_REGISTER structures used by a device will change.

GevGetCameraRegisters

```
GEV_STATUS GevGetCameraRegisters (GEV_CAMERA_HANDLE handle,
                                  DALSA_GENICAM_GIGE_REGS *camera_registers,
                                  int size);
```

Description

Gets the Camera Registers stored with the camera's handle.

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera to be accessed.
**camera_registers*	Pointer to a structure, allocated by the application, to contain the camera registers.
size	Size of the camera registers structure, in bytes.

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_NULL_PTR

GevGetNumberOfRegisters

```
GEV_STATUS GevGetNumberOfRegisters(GEV_CAMERA_HANDLE handle, UINT32 *pNumReg);
```

Description

Gets the number of Camera register entries configured for the camera.
Returns the number of valid GEV_REGISTER structures defined in the camera handle.

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera whose registers are to be accessed.
pNumReg	Pointer to storage to return the number of valid GEV_REGISTER structures in.

Return Value

GEV_STATUS STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_NULL_PTR

GevGetRegisterByName

```
GEV_STATUS GevGetRegisterByName (GEV_CAMERA_HANDLE handle, char *name,
                                 GEV_REGISTER *pReg);
```

Description

Gets a Camera Register structure by name.

This function finds and returns a GEV_REGISTER structure from the camera using the name of the GEV_REGISTER structure. If the name is not matched in the list of registers, an error is returned.

Note: The name is case-sensitive.

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera whose registers are to be accessed.
name	The name to use to search for a GEV_REGISTER structure for the camera.
pReg	Pointer to a GEV_REGISTER data structure, allocated by the application, to contain the GEV_REGISTER data copied from the internal camera configuration data,

Return Value

GEV_STATUS STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_NULL_PTR

GevGetRegisterByIndex

```
GEV_STATUS GevGetRegisterByIndex (GEV_CAMERA_HANDLE handle, UINT32 index,
                                  GEV_REGISTER *pReg);
```

Description

Gets a Camera Register structure by index.

This function finds and returns a GEV_REGISTER structure from the camera using the index of the GEV_REGISTER structure.

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera whose registers are to be accessed.
index	Index to use to access the available GEV_REGISTER structures for the camera.
pReg	Pointer to a GEV_REGISTER data structure, allocated by the application, to contain the GEV_REGISTER data copied from the internal camera configuration data,

Return Value

GEV_STATUS STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_NULL_PTR

GevGetRegisterNameByIndex

```
GEV_STATUS GevGetRegisterNameByIndex(GEV_CAMERA_HANDLE handle, UINT32 index,
                                     int size, char *name)
```

Description

Gets the name of a Camera register entry based on its index.

Returns the name of a GEV_REGISTER structure defined in the camera handle based on the input index.

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera whose registers are to be accessed.
index	Index to use to access the available GEV_REGISTER structures for the camera.
size	Number of bytes available to store the name (should be FEATURE_NAME_MAX_SIZE (48)).
name	Pointer to storage to return the name of the register structure in.

Return Value

GEV_STATUS STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_NULL_PTR

GevGetRegisterPtrByIndex

```
GEV_STATUS GevGetRegisterPtrByIndex (GEV_CAMERA_HANDLE handle, UINT32 index,
                                     GEV_REGISTER **pReg)
```

Description

Gets a pointer to a Camera Register structure by index.

This function finds and returns a pointer to a GEV_REGISTER structure from the camera using the index of the GEV_REGISTER structure.

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera whose registers are to be accessed.
index	Index to use to access the available GEV_REGISTER structures for the camera.
pReg	Pointer to hold a pointer to a GEV_REGISTER data structure, obtained from the internal camera configuration data,

Return Value

GEV_STATUS STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_NULL_PTR

GevGetRegisterPtrByName

```
GEV_STATUS GevGetRegisterPtrByName(GEV_CAMERA_HANDLE handle, char *name,
                                   GEV_REGISTER **pReg)
```

Description

Gets a pointer to a Camera Register structure by name.

This function finds and returns a pointer to a GEV_REGISTER structure from the camera using the name of the GEV_REGISTER structure. If the name is not matched in the list of registers a NULL pointer is returned.

Note: The name is case sensitive.

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera whose registers are to be accessed.
name	Name to use to search for a GEV_REGISTER structure for the camera.
pReg	Pointer to hold a pointer to a GEV_REGISTER data structure, obtained from the internal camera configuration data,

Return Value

GEV_STATUS STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_NULL_PTR

GevInitCameraRegisters

```
GEV_STATUS GevInitCameraRegisters(GEV_CAMERA_HANDLE handle);
```

Description

Initializes Camera Registers.

For supported Teledyne DALSA cameras, this is automatically done when the camera is opened. Users generating their own camera register structure should see ‘cameraregdata.c’ in order to have this function set up their registers automatically.

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera whose registers are to be initialized.

Return Value

GEV_STATUS STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_SOFTWARE (camera registers structure is not properly set up) GEVLIB_ERROR_NULL_PTR

GevReadRegisterByName

```
GEV_STATUS GevReadRegisterByName(GEV_CAMERA_HANDLE handle, char *name, int selector,
                                  UINT32 size, void *data);
```

Description

Reads a camera register, identified by name. A helper function using the pattern GevGetRegisterPtrByName and GevRegisterRead.

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera.
name	Name to use to search for a GEV_REGISTER structure for the camera.
selector	Index into a group of registers providing the same functionality. These register groups need to be set up properly in the GEV_REGISTER structure. This is generally 0 as the ‘array’ based functions can be used to access multiple contiguous locations.
size	Size of the data to be read.
**data*	Pointer to a location, allocated by the caller, to receive the data to be read.

Return Value

GEV_STATUS STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_RESOURCE_NOT_ENABLED
(GEV_REGISTER struct is for a register that is not available)
GEVLIB_ERROR_NOT_IMPLEMENTED (GEV_REGISTER struct is for a Write-Only register)
GEVLIB_ERROR_SOFTWARE (GEV_REGISTER struct defines an unsupported register type)

GevRegisterRead

```
GEV_STATUS GevRegisterRead(GEV_CAMERA_HANDLE handle, GEV_REGISTER *pReg,
                           int selector, UINT32 size, void *data);
```

Description

Reads the specified register (a generic register access function)

Parameters

handle GEV_CAMERA_HANDLE identifying the camera.

**pReg* Pointer to the GEV_REGISTER structure for the register to be accessed.

selector Index into a group of registers providing the same functionality. These register groups need to be set up properly in the GEV_REGISTER structure.

This is generally 0 as the 'array' based functions can be used to access multiple contiguous locations.

size Size of the data to be read.

**data* Pointer to a location, allocated by the caller, to receive the data to be read.

Return Value

GEV_STATUS STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_RESOURCE_NOT_ENABLED
(GEV_REGISTER struct is for a register that is not available)
GEVLIB_ERROR_NOT_IMPLEMENTED (GEV_REGISTER struct is for a Write-Only register)
GEVLIB_ERROR_SOFTWARE (GEV_REGISTER struct defines an unsupported register type)

GevRegisterReadArray

```
GEV_STATUS GevRegisterReadArray (GEV_CAMERA_HANDLE handle, GEV_REGISTER *pReg,
                                 int selector, UINT32 array_offset,
                                 UINT32 num_entries, void *data);
```

Description

Reads an array of 32-bit values from a memory area on the camera.

Parameters

handle GEV_CAMERA_HANDLE identifying the camera.

**pReg* Pointer to the GEV_REGISTER structure for the register to be accessed.

selector Index into a group of registers providing the same functionality. These register groups need to be set up properly in the GEV_REGISTER structure. This is generally 0 for arrays

array_offset Start offset into the array.

num_entries Number of entries to be read from the array, starting at the start offset.

**data* Pointer to a location allocated by the caller, to receive the data read from the array.

Return Value

GEV_STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_RESOURCE_NOT_ENABLED (GEV_REGISTER struct is for a register that is not available)
GEVLIB_ERROR_NOT_IMPLEMENTED (GEV_REGISTER struct is for a Write-Only register)
GEVLIB_ERROR_SOFTWARE (GEV_REGISTER struct does not define an array

GevRegisterReadFloat

```
GEV_STATUS GevRegisterReadFloat (GEV_CAMERA_HANDLE handle, GEV_REGISTER *pReg,
                                 int selector, float *value);
```

Description

Reads a floating point value from a register (a float register access function).

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera.
**pReg*	Pointer to the GEV_REGISTER structure for the register to be accessed.
selector	Index into a group of registers providing the same functionality. These register groups need to be set up properly in the GEV_REGISTER structure. This is generally 0 as the 'array' based functions can be used to access multiple contiguous locations.
value	Pointer to a location to receive the floating point value from the camera.

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_PARAMETER_INVALID (GEV_REGISTER struct is not for an Integer register) GEVLIB_ERROR_ARG_INVALID (GEV_REGISTER definition is invalid) GEVLIB_ERROR_RESOURCE_NOT_ENABLED (GEV_REGISTER struct is for a register that is not available) GEVLIB_ERROR_NOT_IMPLEMENTED (GEV_REGISTER struct is for a Read-Only register) GEVLIB_ERROR_SOFTWARE (GEV_REGISTER struct defines an unsupported register type)

GevRegisterReadInt

```
GEV_STATUS GevRegisterReadInt(GEV_CAMERA_HANDLE handle, GEV_REGISTER *pReg,
                              int selector, UINT32 *value);
```

Description

Reads an integer value from a register (an integer register access function)

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera.
**pReg*	Pointer to the GEV_REGISTER structure for the register to be accessed.
selector	Index into a group of registers providing the same functionality. These register groups need to be set up properly in the GEV_REGISTER structure. This is generally 0 as the 'array' based functions can be used to access multiple contiguous locations.
value	Pointer to a location to receive the integer value from the camera.

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_PARAMETER_INVALID (GEV_REGISTER struct is not for an Integer register) GEVLIB_ERROR_ARG_INVALID (GEV_REGISTER definition is invalid) GEVLIB_ERROR_RESOURCE_NOT_ENABLED (GEV_REGISTER struct is for a register that is not available) GEVLIB_ERROR_NOT_IMPLEMENTED (GEV_REGISTER struct is for a Read-Only register) GEVLIB_ERROR_SOFTWARE (GEV_REGISTER struct defines an unsupported register type)

GevRegisterWrite

```
GEV_STATUS GevRegisterWrite(GEV_CAMERA_HANDLE handle, GEV_REGISTER *pReg,
                            int selector, UINT32 size, void *data);
```

Description

Writes a value to a specified register (a generic register access function)

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera.
**pReg*	Pointer to the GEV_REGISTER structure for the register to be accessed.
selector	Index into a group of registers providing the same functionality. These register groups need to be set up properly in the GEV_REGISTER structure. This is generally 0 as the ‘array’ based functions can be used to access multiple contiguous locations.
size	Size of the data being written.
**data*	Pointer to the data to be written.

Return Value

GEV_STATUS STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_RESOURCE_NOT_ENABLED
(GEV_REGISTER struct is for a register that is not available)
GEVLIB_ERROR_NOT_IMPLEMENTED (GEV_REGISTER struct is for a Read-Only register)
GEVLIB_ERROR_SOFTWARE (GEV_REGISTER struct defines an unsupported register type)

GevRegisterWriteArray

```
GEV_STATUS GevRegisterWriteArray(GEV_CAMERA_HANDLE handle, GEV_REGISTER *pReg,
                                 int selector, UINT32 array_offset,
                                 UINT32 num_entries, void *data);
```

Description

Writes an array of 32-bit values to a memory area on the camera.

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera.
**pReg*	Pointer to the GEV_REGISTER structure for the register to be accessed.
selector	Index into a group of registers providing the same functionality. These register groups need to be set up properly in the GEV_REGISTER structure. This is generally 0.
array_offset	Start offset into the array.
num_entries	Number of entries to be written starting at the start offset.
**data*	Pointer to the data to be written.

Return Value

GEV_STATUS	Possible values are : GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_RESOURCE_NOT_ENABLED (GEV_REGISTER struct is for a register that is not available) GEVLIB_ERROR_NOT_IMPLEMENTED (GEV_REGISTER struct is for a Read-Only register) GEVLIB_ERROR_SOFTWARE (GEV_REGISTER struct does not define an array

GevRegisterWriteFloat

```
GEV_STATUS GevRegisterWriteFloat (GEV_CAMERA_HANDLE handle, GEV_REGISTER *pReg,
                                  int selector, float value);
```

Description

Writes a floating point value to a register (a float register access function)

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera.
**pReg*	Pointer to the GEV_REGISTER structure for the register to be accessed.
selector	Index into a group of registers providing the same functionality. These register groups need to be set up properly in the GEV_REGISTER structure. This is generally 0 as the 'array' based functions can be used to access multiple contiguous locations.
value	Value to be written to the camera.

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_PARAMETER_INVALID (GEV_REGISTER struct is not for an Integer register) GEVLIB_ERROR_ARG_INVALID (GEV_REGISTER definition is invalid) GEVLIB_ERROR_RESOURCE_NOT_ENABLED (GEV_REGISTER struct is for a register that is not available) GEVLIB_ERROR_NOT_IMPLEMENTED (GEV_REGISTER struct is for a Read-Only register) GEVLIB_ERROR_SOFTWARE (GEV_REGISTER struct defines an unsupported register type)

GevRegisterWriteInt

```
GEV_STATUS GevRegisterWriteInt (GEV_CAMERA_HANDLE handle, GEV_REGISTER *pReg,
                                int selector, UINT32 value);
```

Description

Writes an integer value to a register (an integer register access function)

Parameters

handle	identifying the camera.
**pReg*	Pointer to the GEV_REGISTER structure for the register to be accessed.
selector	Index into a group of registers providing the same functionality. These register groups need to be set up properly in the GEV_REGISTER structure. This is generally 0 as the ‘array’ based functions can be used to access multiple contiguous locations.
value	Value to write.

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_PARAMETER_INVALID (GEV_REGISTER struct is not for an Integer register) GEVLIB_ERROR_ARG_INVALID (GEV_REGISTER definition is invalid) GEVLIB_ERROR_RESOURCE_NOT_ENABLED (GEV_REGISTER struct is for a register that is not available) GEVLIB_ERROR_NOT_IMPLEMENTED (GEV_REGISTER struct is for a Read-Only register) GEVLIB_ERROR_SOFTWARE (GEV_REGISTER struct defines an unsupported register type)

GevRegisterWriteNoWait

```
GEV_STATUS GevRegisterWriteNoWait (GEV_CAMERA_HANDLE handle, GEV_REGISTER *pReg,
                                   int selector, UINT32 size, void *data);
```

Description

Writes a value to a register without waiting for an acknowledgment that the write succeeded. (A generic register access function).

Note: Writing without waiting for an ack will queue writes in the camera. Eventually the caller should perform a write with an ack in order to make sure all of the queued writes complete before the queue overflows. The number of writes that can be safely queued is dependent on the camera itself. For Teledyne DALSA cameras, this is typically at least 16 write,

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera.
**pReg*	Pointer to the GEV_REGISTER structure for the register to be accessed.
selector	Index into a group of registers providing the same functionality. These register groups need to be set up properly in the GEV_REGISTER structure. This is generally 0 as the ‘array’ based functions can be used to access multiple contiguous locations.
size	Size of the data being written.
**data*	Pointer to the data being written.

Return Value

GEV_STATUS Possible values are

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE

GEVLIB_ERROR_RESOURCE_NOT_ENABLED
(GEV_REGISTER struct is for a register that is not available)

GEVLIB_ERROR_NOT_IMPLEMENTED
(GEV_REGISTER struct is for a Read-Only register)

GEVLIB_ERROR_SOFTWARE
(GEV_REGISTER struct defines an unsupported register type)

GevSetCameraRegInfo

```
GEV_STATUS GevSetCameraRegInfo (GEV_CAMERA_HANDLE handle, cameraType type,
                                BOOL fSupportedDalsaCamera,
                                DALSA_GENICAM_GIGE_REGS *camera_registers,
                                int size);
```

Description

Sets the Camera Register Info

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera to be accessed.
type	Type of the camera.
fSupportedDalsaCamera	True if the camera is a supported Teledyne DALSA camera.
**camera_registers*	Pointer to the camera registers structure to be assigned to the camera handle,
size	Size of the camera registers structure.

Return Value

GEV_STATUS STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_NULL_PTR

GevWriteRegisterByName

```
GEV_STATUS GevWriteRegisterByName(GEV_CAMERA_HANDLE handle, char *name,
                                  int selector, UINT32 size, void *data);
```

Description

Writes a camera register, identified by name. A helper function using the pattern GevGetRegisterPtrByName and GevRegisterWrite

Parameters

handle	GEV_CAMERA_HANDLE identifying the camera.
name	Name to use to search for a GEV_REGISTER structure for the camera.
selector	Index into a group of registers providing the same functionality. These register groups need to be set up properly in the GEV_REGISTER structure. This is generally 0 as the ‘array’ based functions can be used to access multiple contiguous locations.
size	Size of the data being written.
**data*	Pointer to the data to be written.

Return Value

GEV_STATUS STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_RESOURCE_NOT_ENABLED (GEV_REGISTER struct is for a register that is not available) GEVLIB_ERROR_NOT_IMPLEMENTED (GEV_REGISTER struct is for a Read-Only register) GEVLIB_ERROR_SOFTWARE (GEV_REGISTER struct defines an unsupported register type)

Appendix D: Legacy Functions

This appendix describes legacy GigE-Vision Framework functions that have been replaced but are still supported; new applications should not use these functions.

GevAbortImageTransfer

```
GEV_STATUS GevAbortImageTransfer(GEV_CAMERA_HANDLE handle);
```

Description

Stops the streaming transfer immediately.

Parameters

handle	Handle to the camera

Return Value

GEV_STATUS Possible values are:
GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE (other errors from GevRegisterWriteInt)

Replacement Function

GevAbortTransfer

GevFreeImageTransfer

```
GEV_STATUS GevFreeImageTransfer(GEV_CAMERA_HANDLE handle);
```

Description

Frees a streaming transfer to the list of buffers indicated.

Parameters

handle	Handle to the camera.

Return Value

GEV_STATUS Possible values are:
GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_TIMEOUT (streaming thread did not respond within 5 seconds)

Replacement Function

GevFreeTransfer

GevGetImage

```
GEV_STATUS GevGetImage (GEV_CAMERA_HANDLE handle,
                        GEV_BUFFER_OBJECT **image_object_ptr);
```

Description

Returns the pointer to the next acquired image object acquired images.
If no images are available in the queue, a NULL pointer is returned.

Parameters

handle	Handle to the camera
image_object_ptr	Pointer to receive the image object pointer.

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_TIME_OUT GEVLIB_ERROR_NULL_PTR

Replacement Function

None

GevGetImageBuffer

```
GEV_STATUS GevGetImageBuffer(GEV_CAMERA_HANDLE handle, void **image_buffer_ptr);
```

Description

Returns the pointer to the most recently acquired image buffer data. If no buffer has been acquired, a NULL pointer is returned with a timeout condition.

Parameters

handle	Handle to the camera
image_buffer_ptr	Pointer to receive the image buffer data pointer.

Return Value

GEV_STATUS	Possible values are GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_TIME_OUT GEVLIB_ERROR_NULL_PTR

Replacement Function

None

GevGetImageParameters, GevSetImageParameters

```
GEV_STATUS GevGetImageParameters(GEV_CAMERA_HANDLE handle, PUINT32 width,
                                 PUINT32 height, PUINT32 x_offset,
                                 PUINT32 y_offset, PUINT32 format);

GEV_STATUS GevSetImageParameters(GEV_CAMERA_HANDLE handle, UINT32 width,
                                 UINT32 height, UINT32 x_offset, UINT32 y_offset,
                                 UINT32 format);
```

Description

Gets/sets image parameters from the camera. The current height, width, x/y origin, and image data format can be manipulated with these functions. (Note : Some cameras allow the format of the image data to be changed whereas others do not.)

Parameters

width Image width setting (in pixels).

height Image height setting (in lines).

x_offset Image X (pixel) origin (in pixels).

y_offset Image Y (line) origin (in lines).

format Enumerated value for image format. The value depend on the camera model. Possible values are:

fmtMono8	0x01080001	8 Bit Monochrome Unsigned
fmtMono8Signed	0x01080002	8 Bit Monochrome Signed
fmtMono10	0x01100003	10 Bit Monochrome Unsigned
fmtMono10Packed	0x010C0004	10 Bit Monochrome Packed
fmtMono12	0x01100005	12 Bit Monochrome Unsigned
fmtMono12Packed	0x010C0006	8 Bit Monochrome Packed
fmtMono14	0x01100025	14 Bit Monochrome Unsigned
fmtMono16	0x01100007	16 Bit Monochrome Unsigned
fMtBayerGR8	0x01080008	8-bit Bayer
fMtBayerRG8	0x01080009	8-bit Bayer
fMtBayerGB8	0x0108000A	8-bit Bayer
fMtBayerBG8	0x0108000B	8-bit Bayer
fMtBayerGR10	0x0110000C	10-bit Bayer
fMtBayerRG10	0x0110000D	10-bit Bayer
fMtBayerGB10	0x0110000E	10-bit Bayer
fMtBayerBG10	0x0110000F	10-bit Bayer
fMtBayerGR12	0x01100010	12-bit Bayer
fMtBayerRG12	0x01100011	12-bit Bayer
fMtBayerGB12	0x01100012	12-bit Bayer
fMtBayerBG12	0x01100013	12-bit Bayer
fmtRGB8Packed	0x02180014	8 Bit RGB Unsigned in 24bits
fmtBGR8Packed	0x02180015	8 Bit BGR Unsigned in 24bits
fmtRGBA8Packed	0x02200016	8 Bit RGB Unsigned
fmtBGRA8Packed	0x02200017	8 Bit BGR Unsigned
fmtRGB10Packed	0x02300018	10 Bit RGB Unsigned

fmtBGR10Packed	0x02300019	10 Bit BGR Unsigned
fmtRGB12Packed	0x0230001A	12 Bit RGB Unsigned
fmtBGR12Packed	0x0230001B	12 Bit BGR Unsigned
fmtRGB10V1Packed	0x0220001C	10 Bit RGB custom V1 (32bits)
fmtRGB10V2Packed	0x0220001D	10 Bit RGB custom V2 (32bits)
fmtYUV411packed	0x020C001E	YUV411 (composite color)
fmtYUV422packed	0x0210001F	YUV422 (composite color)
fmtYUV444packed	0x02180020	YUV444 (composite color)
fmtRGB8Planar	0x02180021	RGB8 Planar buffers
fmtRGB10Planar	0x02300022	RGB10 Planar buffers
fmtRGB12Planar	0x02300023	RGB12 Planar buffers
fmtRGB16Planar	0x02300024	RGB16 Planar buffers

Return Value

GEV_STATUS Possible values are:
GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_PARAMETER_INVALID
(GEV_REGISTER struct is not for an Integer register)
GEVLIB_ERROR_ARG_INVALID (GEV_REGISTER definition is invalid)
GEVLIB_ERROR_SOFTWARE
(GEV_REGISTER struct defines an unsupported register type)

Replacement Function

GevGetPayloadParameters

GevGetNextImage

```
GEV_STATUS GevGetNextImage(GEV_CAMERA_HANDLE handle,
                           GEV_BUFFER_OBJECT **image_object_ptr,
                           struct timeval *pTimeout);
```

Description

Waits for the next image object to be acquired and returns its pointer. If no buffer has been acquired before the timeout period expires, a NULL pointer is returned.

Parameters

handle Handle to the camera

image_object_ptr Pointer to receive the image object pointer.

pTimeout Pointer to a struct timeval (microsecond precision) for the timeout period to wait for the next frame.

Return Value

GEV_STATUS Possible values are:
GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_TIME_OUT
GEVLIB_ERROR_NULL_PTR

Replacement Function

GevGetNextFrame

GevInitializeImageTransfer

```
GEV_STATUS GevInitializeImageTransfer (GEV_CAMERA_HANDLE handle, UINT32 numBuffers,
                                       UINT8 **bufAddress);
```

Description

Initializes a streaming transfer to the list of buffers indicated.
The transfer is set up with the Asynchronous cycling mode.

Parameters

handle	Handle to the camera.
numBuffers	Number of buffers addresses in array.
bufAddress	Array of buffer addresses (already allocated).

Return Value

GEV_STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_PARAMETER_INVALID
(GEV_REGISTER struct is not for an Integer register)
GEVLIB_ERROR_ARG_INVALID
(GEV_REGISTER definition is invalid)
GEVLIB_ERROR_SOFTWARE
(GEV_REGISTER struct defines an unsupported register type)

Note: Errors include attempting to initialize the transfer on a connection that is not set up for streaming.

Replacement Function

GevInitializeTransfer

GevInitImageTransfer

```
GEV_STATUS GevInitImageTransfer (GEV_CAMERA_HANDLE handle, GevBufferCyclingMode mode,
                                 UINT32 numBuffers, UINT8 **bufAddress);
```

Description

Initializes a streaming transfer to the list of buffers indicated. The buffer cycling mode is also set.

Parameters

handle Handle to the camera.

mode Buffer cycling mode. Can be either :

Asynchronous: All buffers available all the time with no protection between the application and the acquisition process.

Or

SynchronousNextEmpty; Buffers obtained by the application are available only to the application until released back to the acquisition process. Buffers are filled in the order they are released back to the acquisition process. If there are no more buffers available to the acquisition process, subsequent images are not stored to memory and are deemed to have been sent to the "trash".

numBuffers Number of buffers addresses in array.

bufAddress Array of buffer addresses (already allocated).

Return Value

GEV_STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_PARAMETER_INVALID
(GEV_REGISTER struct is not for an Integer register)
GEVLIB_ERROR_ARG_INVALID
(GEV_REGISTER definition is invalid)
GEVLIB_ERROR_SOFTWARE
(GEV_REGISTER struct defines an unsupported register type)

Note: Errors include attempting to initialize the transfer on a connection that is not set up for streaming.

Replacement Function

GevInitializeTransfer

GevQueryImageTransferStatus

```
GEV_STATUS GevQueryImageTransferStatus(GEV_CAMERA_HANDLE handle,
                                       PUINT32 pTotalBuffers, PUINT32 pNumUsed,
                                       PUINT32 pNumFree, PUINT32 pNumTrashed,
                                       GevBufferCyclingMode *pMode);
```

Description

Releases an image object back to the acquisition process for re-use. It is mandatory to call this function for a transfer using the SynchronousNextEmpty cycle mode in order to avoid running out of images for the acquisitions process to fill. It is not necessary to call this function for a transfer using the Asynchronous cycle mode..

Parameters

handle	Handle to the camera
pTotalBuffers	Pointer to receive the total number of buffers in the transfer list.
pNumUsed	Pointer to receive the number of filled buffers ready to be received from the transfer list.
pNumFree	Pointer to receive the number of empty (free) buffers that are available to be filled.
pNumTrashed	Pointer to receive the total number of buffers that have been “trashed” so far. (i.e. Frames that are dropped when there are no more empty buffers to fill but image data has still been received).
pMode	Pointer to receive the current buffer cycling mode (Asynchronous=0, SynchronousNextEmpty=1).

Return Value

GEV_STATUS	Possible values are: GEVLIB_OK GEVLIB_ERROR_INVALID_HANDLE GEVLIB_ERROR_PARAMETER_INVALID GEVLIB_ERROR_ARG_INVALID

Replacement Function

GevQueryTransferStatus

GevReleaseImage

```
GEV_STATUS GevReleaseImage(GEV_CAMERA_HANDLE handle,
                           GEV_BUFFER_OBJECT **image_object_ptr);
```

Description

Releases an image object back to the acquisition process for re-use. It is mandatory to call this function for a transfer using the SynchronousNextEmpty cycle mode in order to avoid running out of images for the acquisitions process to fill. It is not necessary to call this function for a transfer using the Asynchronous cycle mode..

Parameters

handle	Handle to the camera
image_object_ptr	Pointer to the image object begin released.

Return Value

GEV_STATUS Possible values are:
GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_PARAMETER_INVALID
GEVLIB_ERROR_ARG_INVALID

Replacement Function

GevReleaseFrame

GevReleaseImageBuffer

```
GEV_STATUS GevReleaseImageBuffer(GEV_CAMERA_HANDLE handle, void **image_buffer_ptr);
```

Description

Releases an image object back to the acquisition process for re-use. The image object is identified from the image buffer pointer passed in to the function. It is mandatory to call this function for a transfer using the SynchronousNextEmpty cycle mode in order to avoid running out of images for the acquisition process to fill. It is not necessary to call this function for a transfer using the Asynchronous cycle mode..

Parameters

handle	Handle to the camera
image_buffer_ptr	Pointer to the image buffer data for the image object being released,.

Return Value

GEV_STATUS Possible values are

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_PARAMETER_INVALID
GEVLIB_ERROR_ARG_INVALID

Replacement Function

GevReleaseFrameBuffer

GevStartImageTransfer

```
GEV_STATUS GevStartImageTransfer(GEV_CAMERA_HANDLE handle, UINT32 numFrames);
```

Description

Starts the streaming transfer.

Parameters

handle	Handle to the camera
numFrames	Number of frames to be acquired (-1 for continuous).

Return Value

GEV_STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEV_STATUS_BUSY (camera is busy reconfiguring – try again later)

Replacement Function

GevStartTransfer

GevStopImageTransfer

```
GEV_STATUS GevStopImageTransfer(GEV_CAMERA_HANDLE handle);
```

Description

Stops the streaming transfer.

Parameters

handle	Handle to the camera

Return Value

GEV_STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE (other errors from GevRegisterWriteInt)

Replacement Function

GevStopTransfer

GevWaitForNextImage

```
GEV_STATUS GevWaitForNextImage (GEV_CAMERA_HANDLE handle,
                                GEV_BUFFER_OBJECT **image_object_ptr, UINT32
                                timeout);
```

Description

Waits for the next image object to be acquired and returns its pointer. If no buffer has been acquired before the timeout period expires, a NULL pointer is returned.

Parameters

handle	Handle to the camera
image_object_ptr	Pointer to receive the image object pointer.
timeout	Timeout period (in msec) to wait for the next frame.

Return Value

GEV_STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_TIME_OUT
GEVLIB_ERROR_NULL_PTR

Replacement Function

GevWaitForNextFrame

GevWaitForNextImageBuffer

```
GEV_STATUS GevWaitForNextImageBuffer(GEV_CAMERA_HANDLE handle,
                                     void **image_buffer_ptr, UINT32 timeout);
```

Description

Waits for the next image to be acquired and returns the pointer to the image data. If no buffer has been acquired before the timeout period expires, a NULL pointer is returned.

Parameters

handle	Handle to the camera
image_buffer_ptr	Pointer to receive the image buffer data pointer.
timeout	Timeout period (in msec) to wait for the next.

Return Value

GEV_STATUS Possible values are:

GEVLIB_OK
GEVLIB_ERROR_INVALID_HANDLE
GEVLIB_ERROR_TIME_OUT
GEVLIB_ERROR_NULL_PTR

Replacement Function

None

Contact Information

The following sections provide sales and technical support contact information.

Sales Information

Visit our web site:	www.teledynedalsa.com/corp/contact/
Email:	mailto:info@teledynedalsa.com

Technical Support

Submit any support question or request via our web site:

Technical support form via our web page:	
Support requests for imaging product installations	http://www.teledynedalsa.com/imaging/support
Support requests for imaging applications	
Camera support information	
Product literature and driver updates	